## Data:

The collection of small fact and figures we call it as “data”.

**Example:** Student_id, Student_name, Student_age etc….

---

## Facts:

Essentially it is a specific details like a person's name and age, which fall under the category of data, but they hold significance and relevance solely to that individual. **Example:** Person_name, Person_age etc.

---

## Figures:

It represents visual presentations of data, taking the form of graphs, charts, drawings, image, or maps. They serve to enhance visual impact and effectively communicate the key findings. **Example:** Height of a person, picture of a person etc.

---

## Database:

The collection of large fact and figures we call it as “database”.

**Example:** University details, Bank details, Hospital details etc.

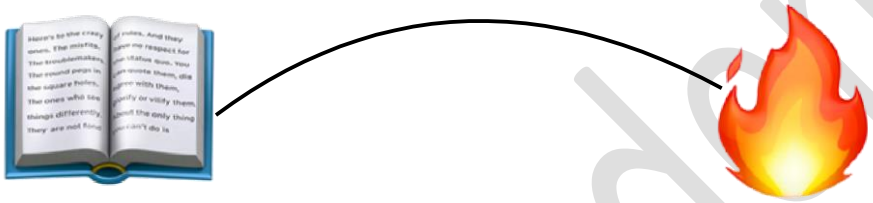
---

Tap Academy

# How Data used to store in the olden days:

Traditionally databases where always stored physically using books called as Registers or they are also called as ledgers. In Big companies these registers where further stored in warehouses. The problem with this was:

- Difficult to store and retrieve data
- Data security was an issue because it could be affected by fire, pests etc.
- Data cannot be easily backed up and hence can be permanently lost.



To overcome all these issues, we decided to store databases within computers. Computers can store large quantity of data and this data can beeasily store and retrieved. Also, data is secure and protected because data can be easily backed up using hard disks and cloud storage technologies.

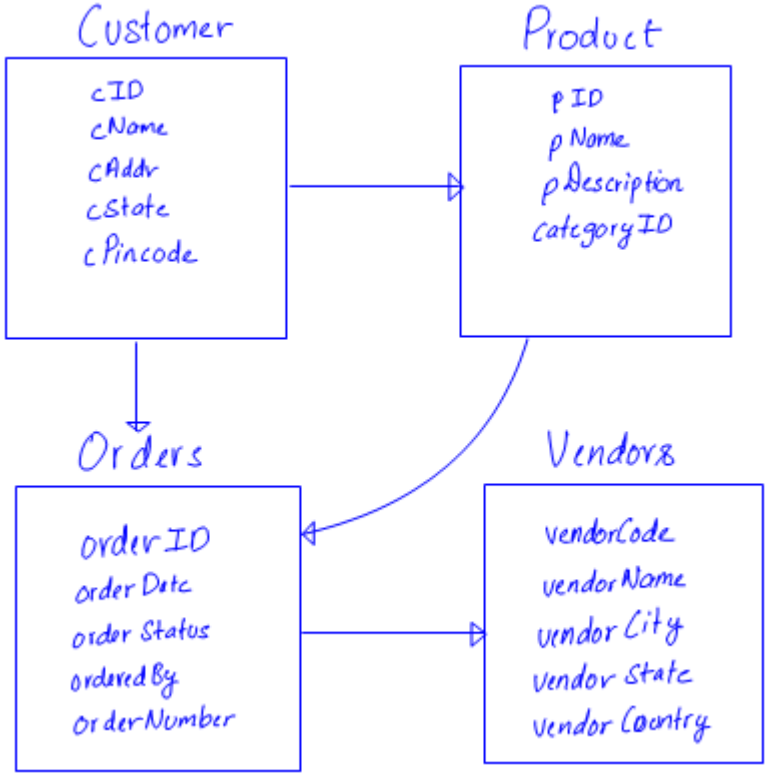
**Computer** **Hard Disk**

Tap Academy

## Data Complexity:

In the real-world data is extremely complex and many a times there are many objects involved each with its own set of data and many of these objects and their data are also interrelated.

Let's take an example of the data related to an amazon website:




In order to store this complex real-world data which is related to eachother efficiently in the computer Dr. Edgar F Codd while working in IBM released a paper on relational database model which from 1960's till date is being used to create and store relational data within database. These databases are called as Relational Databases.

# DBMS:

DBMS stands for Database Management System. It is a software application or system that enables users to interact with a database, providing a set of tools and functionalities to create, update, retrieve, and manage data efficiently and securely.

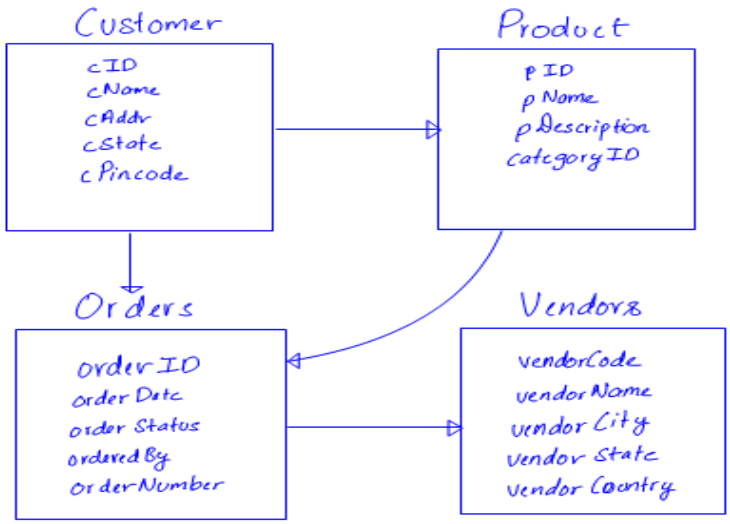
The primary purpose of a DBMS is to act as an intermediary between users and the database, ensuring data integrity, security, and organization.

As students, we should prefer MySQL among the various available DBMS software options like Oracle, Sybase, etc., since many companies and startups also utilize MySQL, and it is a free software.

# Note:

1. Under the DBMS there are lot software available such as MySQL, Oracle, Sybase, DB2 etc.
2. If MySQL or Oracle is used then we need follow some set of rules such as:
    - Table should be created
    - Should specify the column names and datatypes
    - Should insert the data into the table.
3. In MySQL the facts and figures we will be segregating and will keep in different tables and each table will be having a relationship between them as show below:

Tap Academy




4. Because of this reason MySQL, Oracle all this this software's comes under the RDBMS where we will be managing our database but relationship also will be created.

---

## RDBMS:

1. RDBMS stands for Relational Database Management System/Software.
2. It is a type of database management system that is based on the relational model, which was introduced by Dr. Edgar F. Codd in 1970.
3. An RDBMS organizes and stores data in the form of tables with rows and columns, allowing for efficient and structured data management.

---

Tap Academy

# MySQL:

1. MySQL is an open-source relational database management system (RDBMS) that is widely used for managing and organizing data in various applications.
2. It is one of the most popular database systems in the world and is commonly used in web development, data-driven applications, and other software projects.

# History Of MySQL:

1. MySQL was created by a Swedish company, MySQL AB, founded by Swedes David Axmark, Allan Larsson and Finland Swede Michael "Monty" Widenius's.
2. Original development of MySQL by Widenius's and Axmark began in 1994.
3. The first version of MySQL appeared on 23 May 1995.
4. MySQL is named after co-founder Monty Widenius's daughter,
5. My.
6. The recent version of MySQL is 8.0

---

# SQL:

1. SQL stands for Structured Query Language.
2. SQL (Structured Query Language) is a query language used for managing and manipulating relational databases.
3. It allows users to interact with databases to perform various operations, such as data retrieval, insertion, deletion, and modification.
4. SQL is widely used across various industries and applications due to its simplicity, flexibility, and the ability to handle vast amounts of data efficiently.

Tap Academy

5. If we have to communicate with our MySQL database, we have to use SQL.

## History of SQL:

1. SQL was first developed in the early 1970s at IBM by a team led by Donald D. Chamberlin and Raymond F. Boyce.
2. The original name of the language was SEQUEL, which stood for "Structured English Query Language."
3. SEQUEL was later renamed to SQL.
4. SQL quickly became the standard language for interacting with relational databases, and it was soon adopted by other RDBMS vendors such as MySQL, Oracle, Sybase, and Microsoft.

---

Tap Academy

# ER Diagram

- ER Diagram is a diagrammatic representation of a data.
- It stands for Entity Relationship Diagram.

**Entity:** Anything that exist in the real world we call it as an entity.

**Example:** Book, Pen, Laptop, Mobile Phone, Student, Course, Department etc.

**Represented by:**

**Rectangular**

## Types of Entities:

- **Physical Entity (Tangible Entity)**
- **Conceptual Entity (Intangible Entity)**

**Physical Entity:** Anything or object that exist physically in the real world we call it as physical entity.

**Example:** Student, Book, Pen, Laptop etc.

**Conceptual Entity:** Anything or object that does not exist physically in the real world we call it as conceptual entity.

**Example:** Course, Department, Semester, Internet etc.

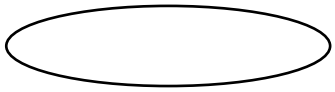
**Note:** Both physical and conceptual entities are represented by rectangular.

Tap Academy

**Attribute:** A property that describes an entity we call it as attribute.

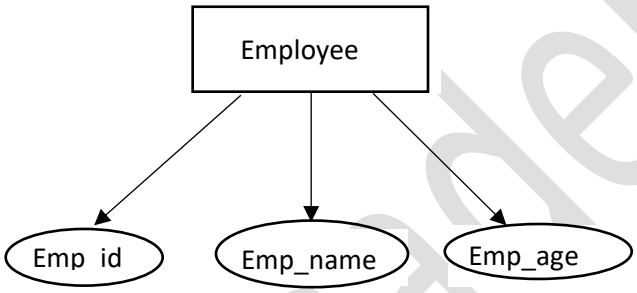
**Example:** id, age, gender, email, phone_number etc.

**Represented by:**



**Ellipse**

**Example:**



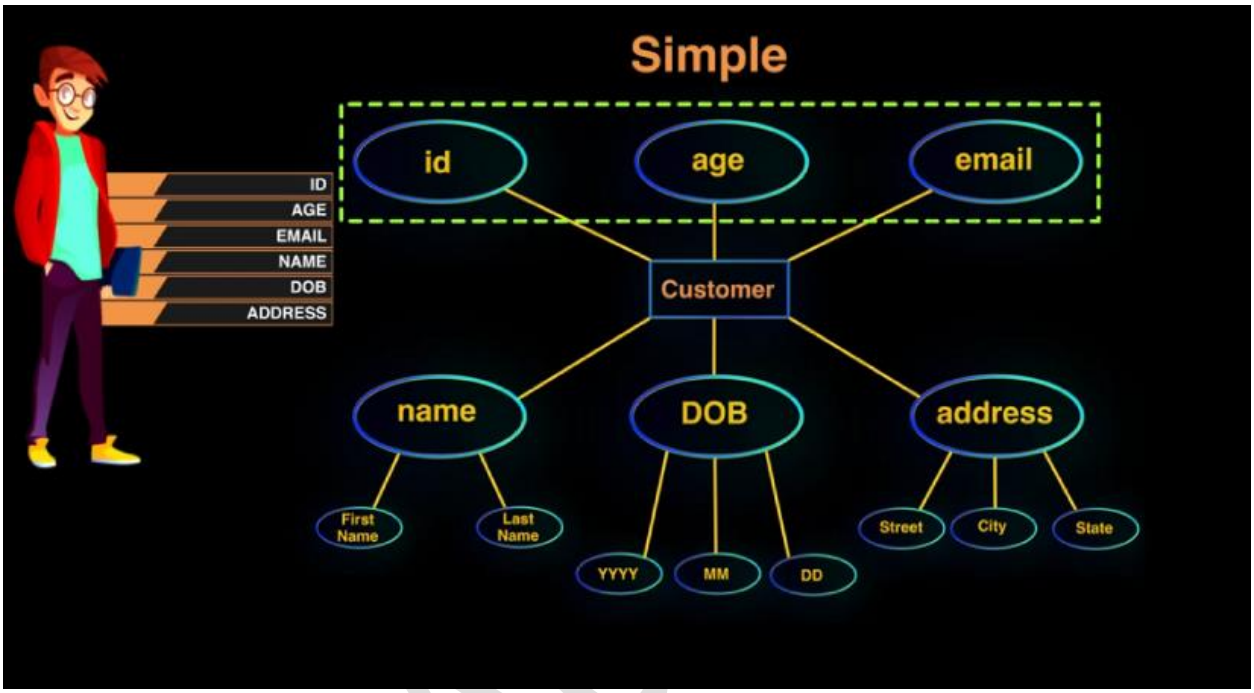

## Types of attributes

- Simple attribute
- Composite attribute
- Single-valued attribute
- Multi-valued attribute
- Derived attribute
- Key attribute

Tap Academy

**Simple Attribute:** Any attributes which cannot be divided further we call them as simple attributes.

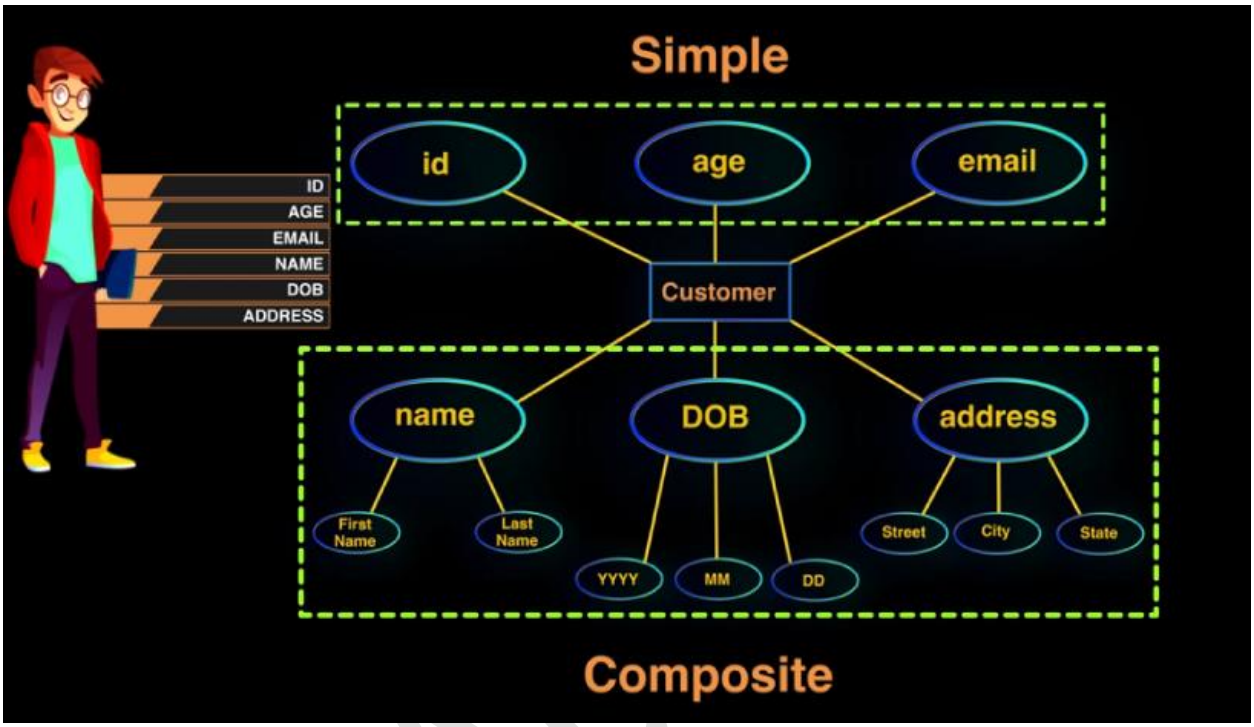
**Example:** id, age, email etc.




Tap Ac

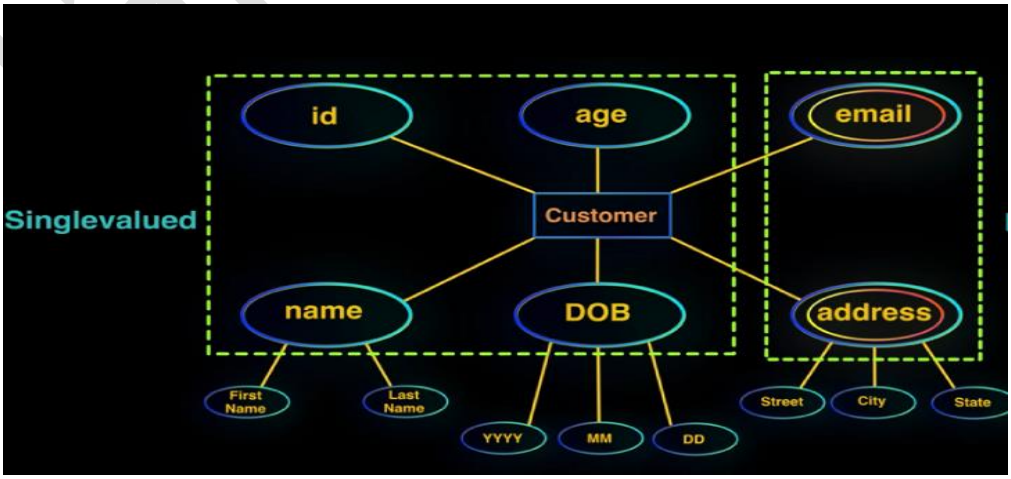
**Composite Attribute:** Any attributes which can be divided further we call them as composite attributes.

**Example:** name, DOB, address etc.,



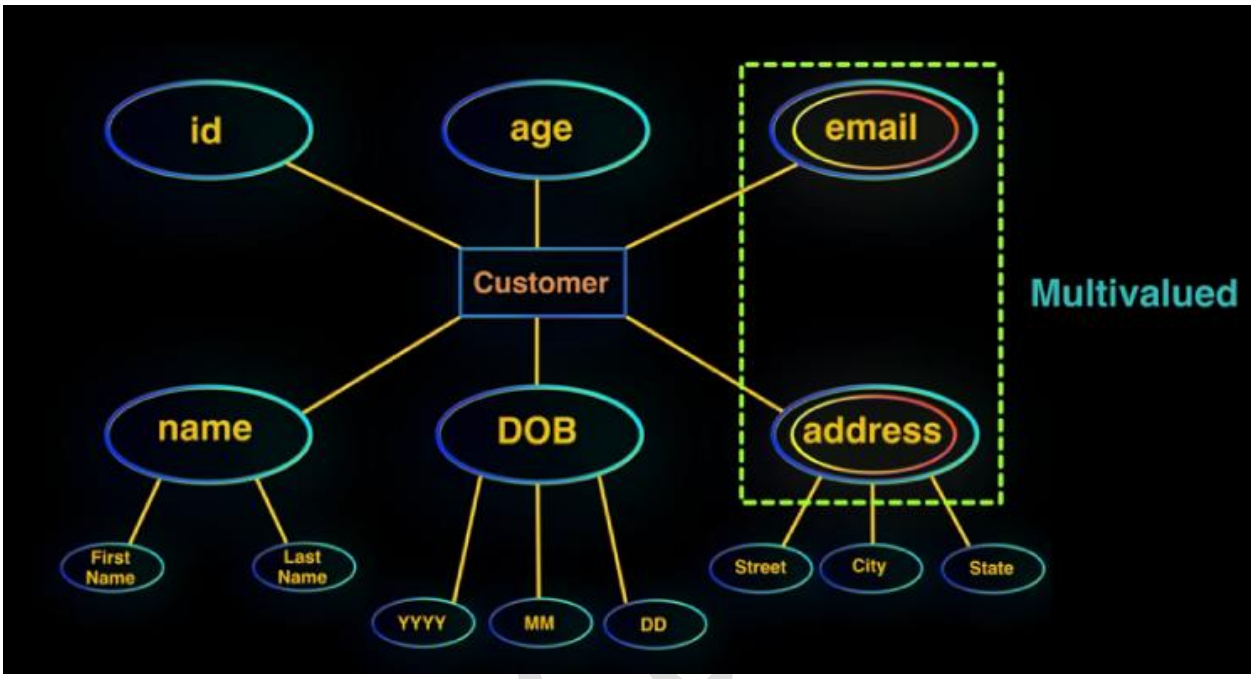

**Single-valued Attribute:** Any attributes which contains only single values we call it as single-valued attribute.

**Example:** id, name etc.,



**Multi-valued Attribute:** Any attributes which can contain multiple values we call it as multi-valued attribute.

**Example:** email, address etc.,



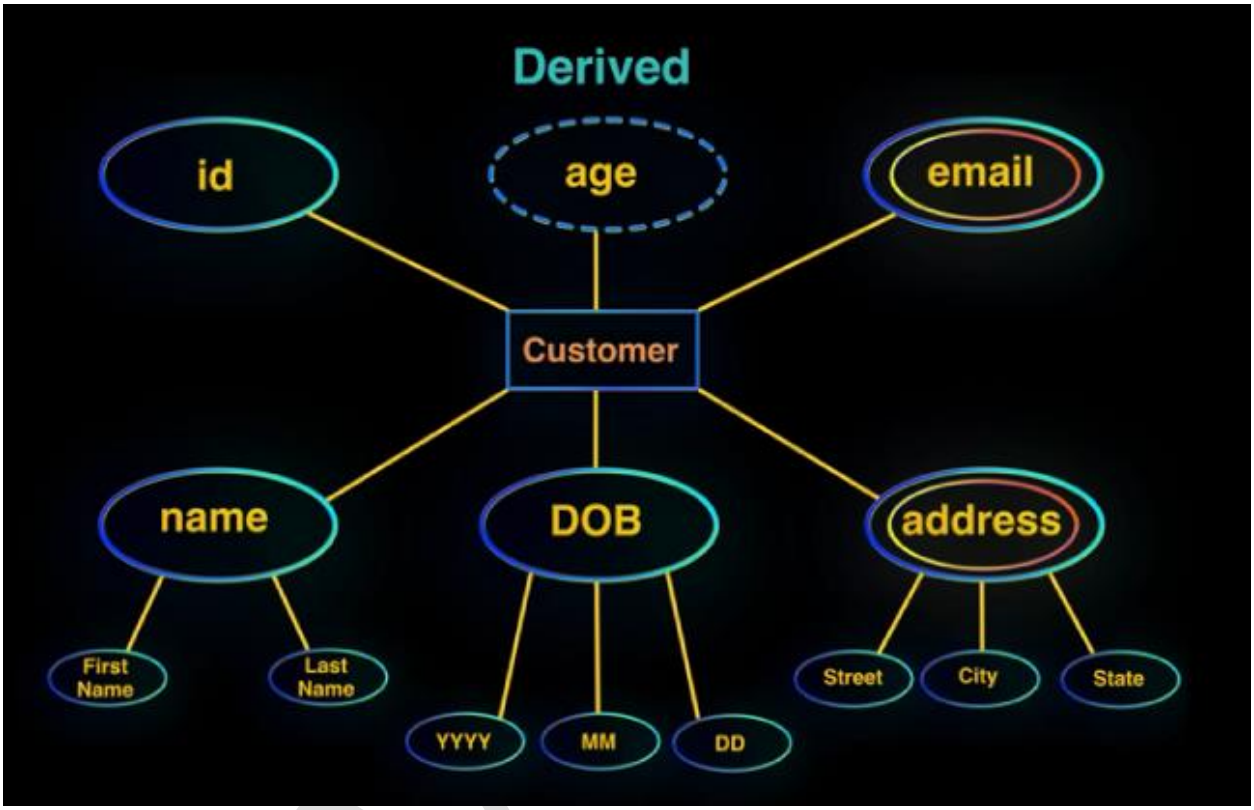

## Note:

- Single-valued attribute is represented by ellipse.
- Multi-valued attribute is represented by double ellipse.

Tap Ac

**Derived Attribute:** Any attribute which can be derived by another attribute we can it as derived attribute like age attribute can be derived by dob attribute hence age attribute, we call it as derived attribute.

**Example:** age etc.,

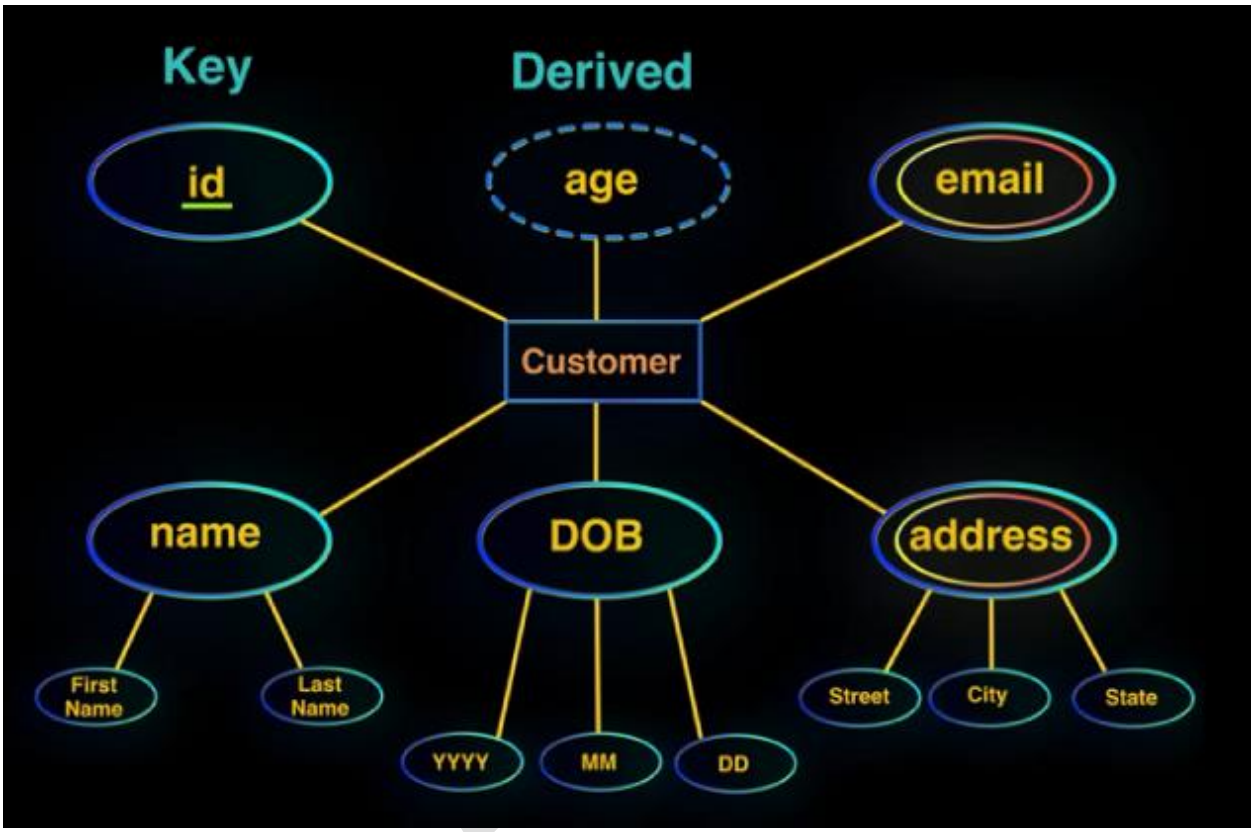



## Note:

- Derived attribute is represented by dotted/dashed ellipse.

**Key Attribute:** Any attribute which can be uniquely identified we call it as key attribute.

**Example:** id etc.,

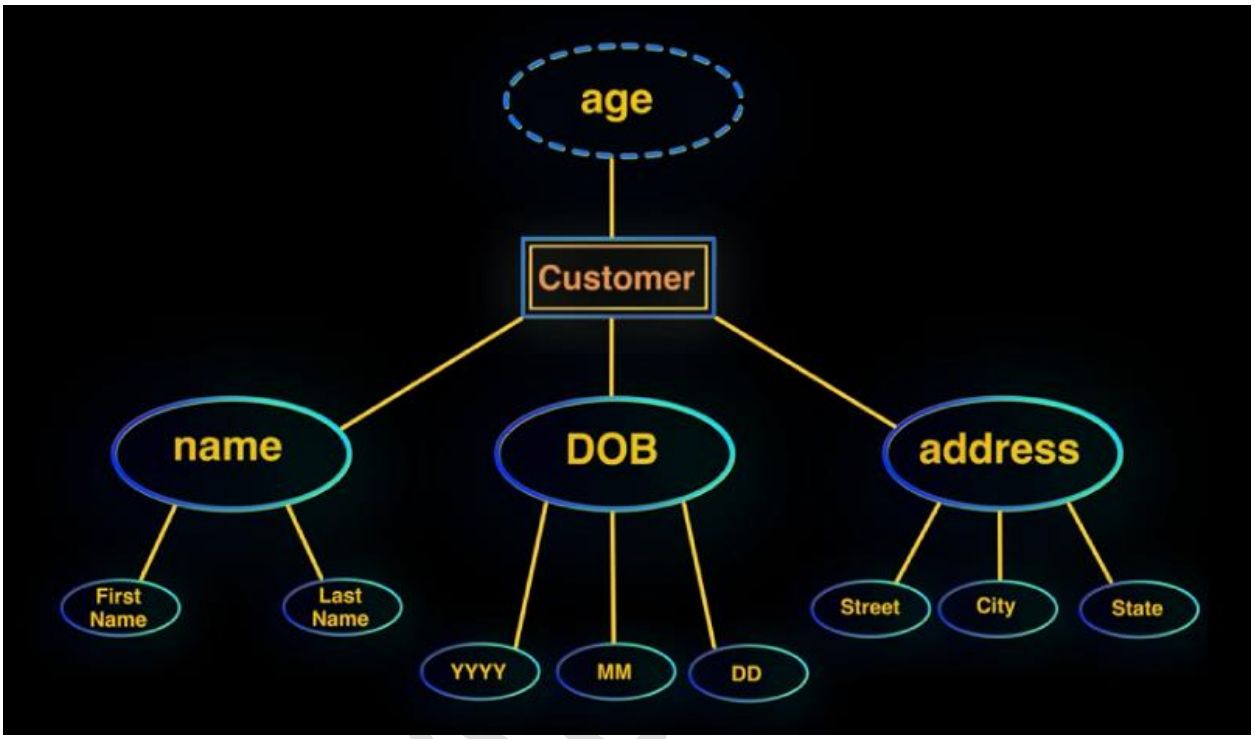



## Note:

- Key attribute is represented by Underline.

# Weak Entity

If an entity does not contain an attribute which can be uniquely identified, then such entities we call it as weak entity.

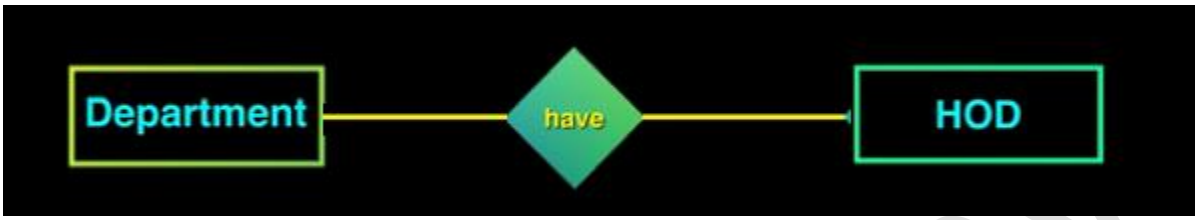



## Note:

- Weak entity is represented by double rectangular.
- Make sure every entity should be a strong entity means every entity should contain atleast one key attribute.

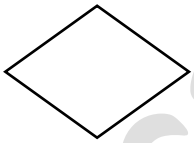
**Relationship:** Association among two entities we call it as “Relationship”.

**Example:** teaches, purchase, works, have etc.




Here department is one entity and HOD is another entity, we are associating both the entities any creating the relationship.

**Represented by:**



**Diamond**

Tap Academy

# Cardinality Ratio or Type of Relationship:

1:1

1:M

M:1

M:N

**Note:** In the ER Diagram just giving the relationship is not sufficient but also, we need to provide what type of relationship an entity contains as show below:

- **1:1**

Here Department and HOD entities have one to one relationship i.e. for one department we will have only one HOD.

Tap Academy

## ➢ 1:M

Here Company and Employee entities have one to many relationship i.e., a one company can have many employees

## ➢ M:1

Here Employee and Company entities have many to one relationship i.e., many employees can work in one company.

## ➢ M:N

Here Product and Customer entities contains many to many relationship, we need to identify the many to many relationship by check in both the directions i.e., we need to get one to many relationship in each direction like one product can be purchased by multiple customers and one customer can purchase multiple products hence we can conclude that product and customer has many to many relationship.

# Scenario:

A university has multiple departments, each with a unique department code and department name. Each department has multiple faculty members, each with a unique faculty ID, name, and email address. Faculty members can take multiple courses, and each course has a unique course code, course title, and credit value. Each course is offered in a specific semester, and each semester has a unique semester code, semester_startdate, semester_enddate. Each semester has multiple students with unique student ID, name, contact number, email ID.

## Attributes

**Departments:**
Each department has a unique department code that serves as its identifier
Each department also has a department name that describes the area of study for that department

**Faculty Members:**
Each faculty member has a unique faculty ID that serves as their identifier
Each faculty member also has a name and an email address that can be used to contact them

**Courses:**
Each course has a unique course code that serves as its identifier
Each course also has a course title that describes the subject matter of the course
Each course has a credit value that represents the amount of credit a student earns upon completion of the course

**Semesters:**
Each semester has a unique semester code that serves as its identifier
Each semester has a start date and an end date that specify the time period during which the semester takes place

**Students:**
Each student has a unique student ID that serves as their identifier
Each student also has a name, contact number, and email ID that can be used to contact them

## Relationships:

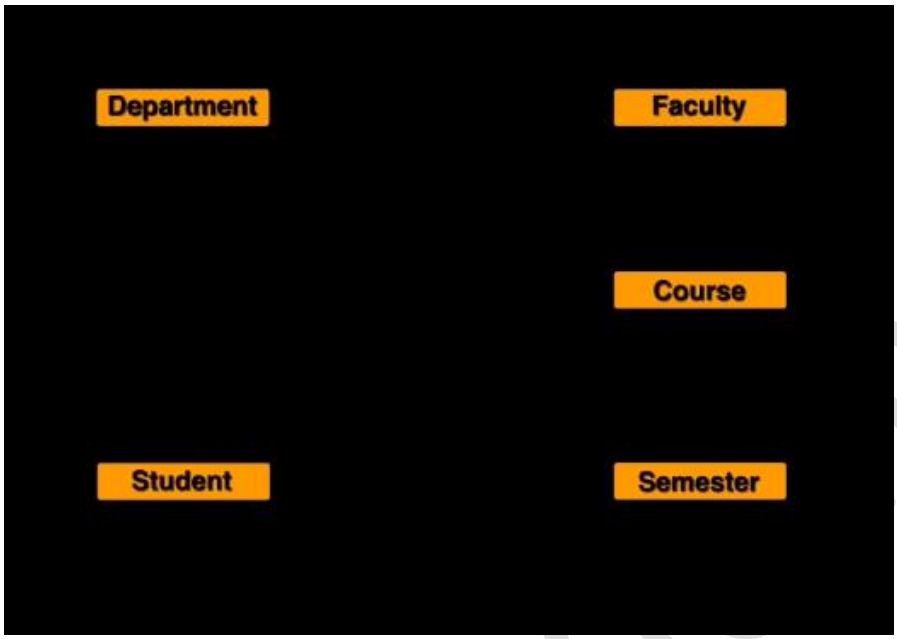
A department has multiple faculty members, and a faculty member belongs to one department (one-to-many relationship)
A faculty member can take multiple courses, and a course can be taught by multiple faculty members (many-to-many relationship)
A course is offered in a specific semester, and a semester can offer multiple courses (many-to-one relationship)
A semester has multiple students, and a student can be enrolled in one semester (one-to-many relationship)
A course has multiple students, and a student can enroll in multiple courses (many-to-many relationship)

# Steps to Draw the ER Diagram

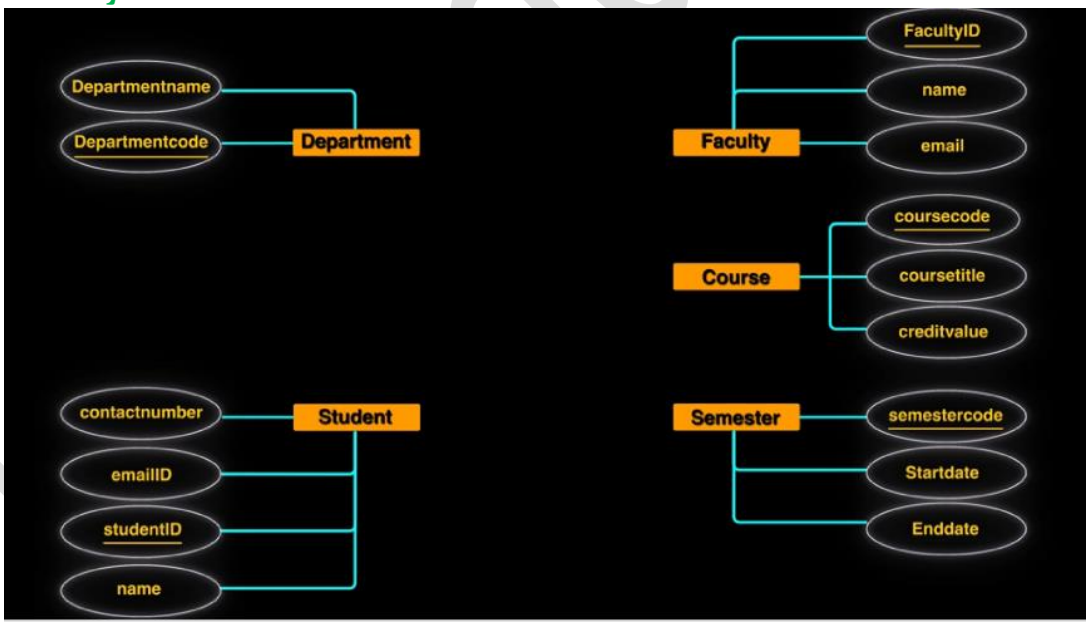
1. Identify the Entities
2. Identify the Attributes
3. Identify the Relationship
4. Identify the Cardinality Ratio

Tap Academy

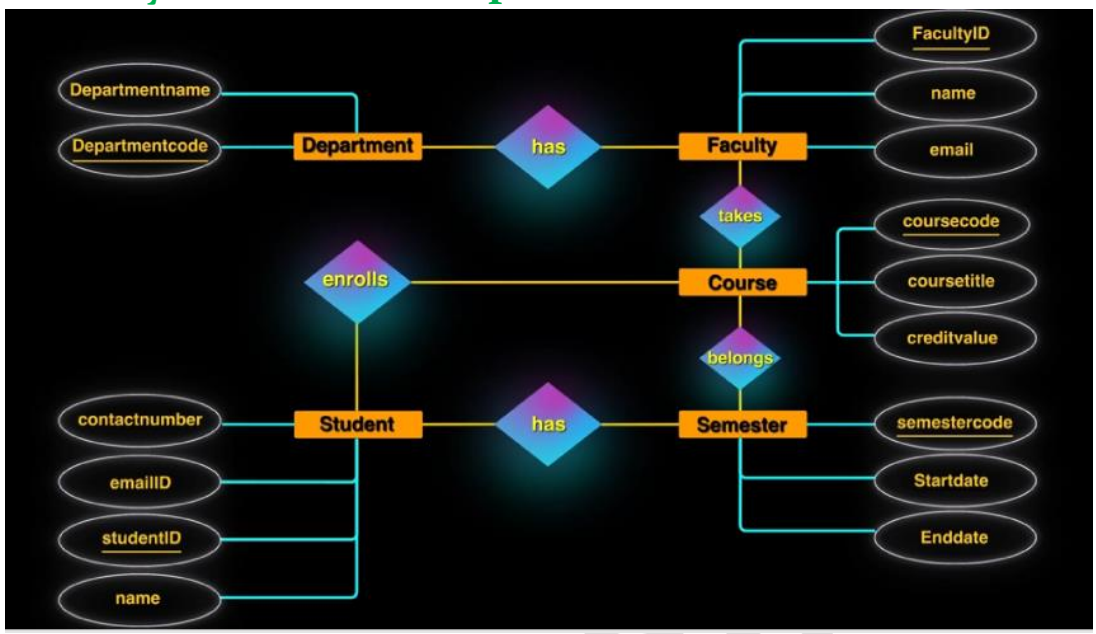
## ➢ Identify the Entities



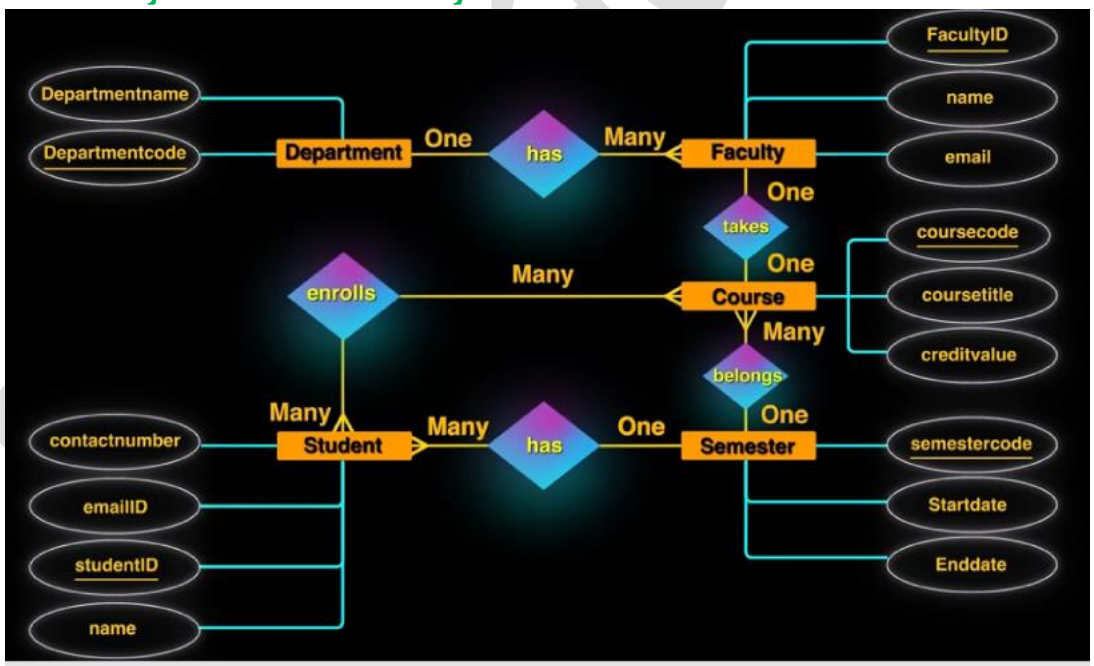

## ➢ Identify the Attributes



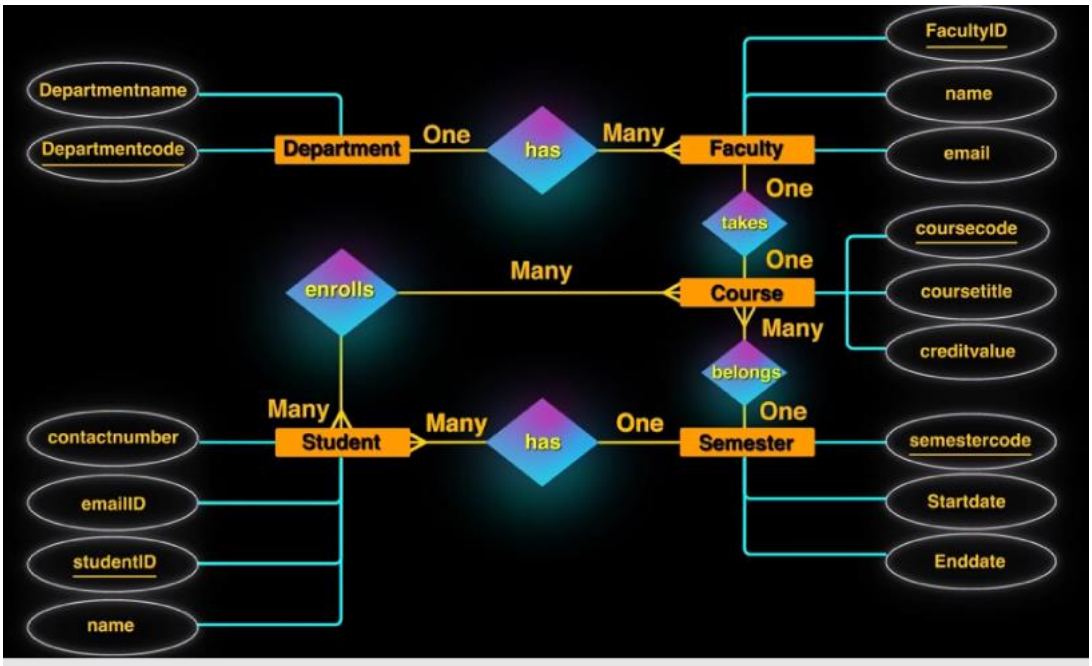
## ➢ Identify the Relationship




## ➢ Identify the cardinality ratio



# The Final ER Diagram for the University Database:



# Converting ER Diagram to ER Schema or Relational Schema.

In SQL we will be storing the data in the table format which contains rows and columns.

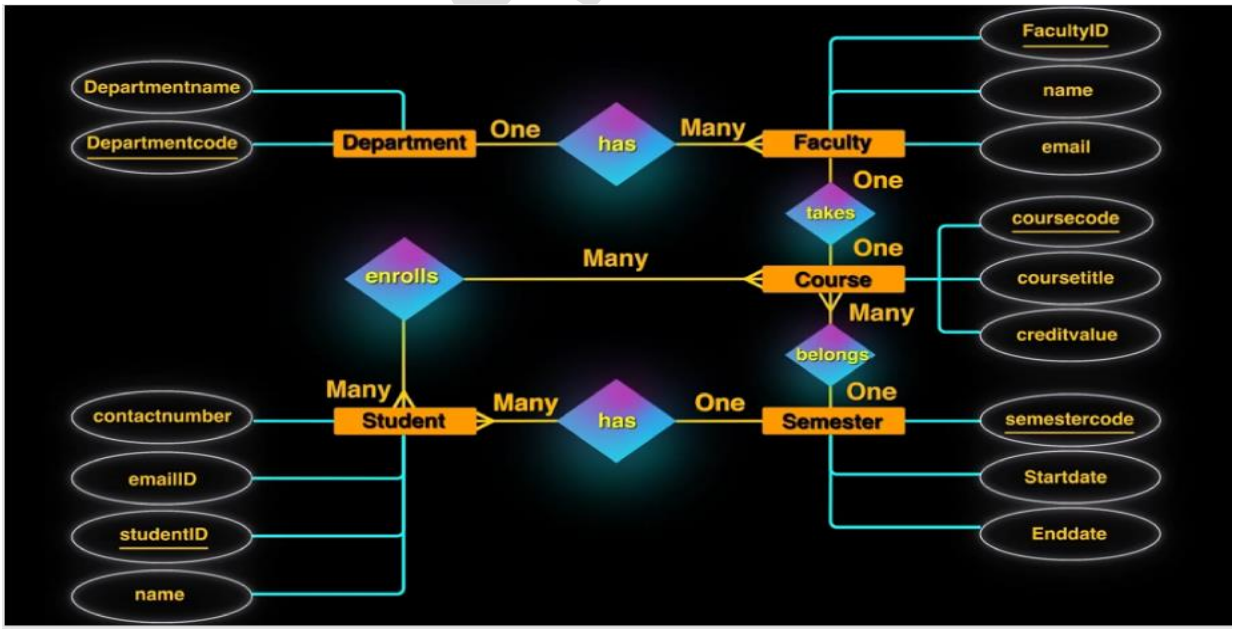
Schema is nothing but it's a collection of tables, which we can also call as database.

So now it's time to convert the ER Diagram into tables.

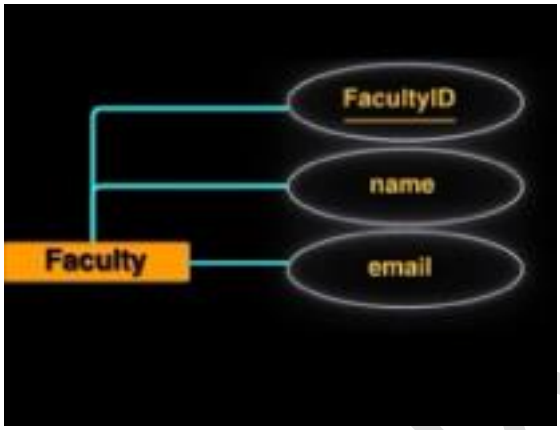
## Note:

1. As we will be having multiple tables, if we want to perform any task on the tables, we have to use the proper table, hence it's important to give unique names to the tables and here entities we have to use as a table names.
2. Attributes should be used as Column names.

Here is the ER Diagram which we will be converting into Relational Schema.



- Lets take one faculty entity first and convert it into the table.




FACULTY

| FacultyID (PK) | name | email |
|---|---|---|
| 91 | Rohit | rohit@gmail.com |
| 92 | Somanna | Somanna@gmail.com |
| 93 | Rakshit | rakshit@gmail.com |
| 94 | Asad | asad@gmail.com |
| 95 | Shilpa | Shilpa@gmail.com |
| 96 | Megha | megha@gmail.com |

Tap Academy

## Note:

The random data is taken inside the tables, just focus on creating the tables with table name and columns, along with the primary key.

## What is Primary key?

- Any attribute which should contain the unique values, we can make it as primary key, if we make any attribute as a primary key then that particular column should not contain any duplicate values or null values.
- Primary key is represented by underline.
- We can have only one primary key in one table.

In the similar manner all the entities and attributes will be converted like shown below from the given ER Diagram:

**DEPARTMENT**

| Departmentcode (PK) | Departmentname |
|---|---|
| 201 | CS |
| 202 | ECE |
| 203 | Mechanical |

**COURSE**

| coursecode (PK) | coursetitle | creditvalue |
|---|---|---|
| 101 | Dynamics | 4 |
| 102 | Fluid Mechanics | 3 |
| 103 | Java | 5 |
| 104 | C++ | 3 |
| 105 | Circuit Analysis | 2 |
| 106 | Digital electronics | 4 |
| 107 | Python | 2 |

**SEMESTER**

| semestercode (PK) | Startdate | Enddate |
|---|---|---|
| 001 | 01/06/2002 | 25/12/2002 |
| 002 | 02/02/2003 | 30/07/2003 |
| 003 | 01/09/2003 | 30/03/2004 |
| 004 | 01/05/2004 | 30/11/2004 |

**STUDENT**

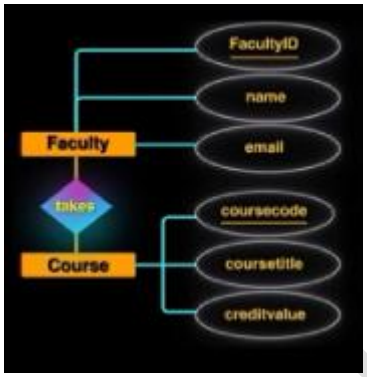
| studentID (PK) | name | email | contactnumber |
|---|---|---|---|
| 301 | Rahul | rahul@gmail.com | 7816529144 |
| 302 | Krishna | krishna@gmail.com | 7895008587 |
| 303 | Sana | sana@gmail.com | 8073409794 |
| 304 | Savita | savita@gmail.com | 6363701421 |
| 305 | Satish | satish@gmail.com | 9620855028 |
| 306 | Divya | divya@gmail.com | 7338573928 |
| 307 | Akash | akash@gmail.com | 7022726043 |

Tap Academy

## Note:

We have seen how to convert the ER diagram to tables, but it's also important to give the relationship between the tables.

1. **First here we are taking faculty and course entity and creating one to one relationship here.**




**COURSE**

| coursecode (PK) | coursetitle | creditvalue | FacultyID (FK) |
|---|---|---|---|
| 101 | Dynamics | 4 | 91 |
| 102 | Fluid Mechanics | 3 | 92 |
| 103 | Java | 5 | 93 |
| 104 | C++ | 3 | 94 |
| 105 | Circuit Analysis | 2 | 95 |
| 106 | Digital electronics | 4 | 96 |

**FACULTY**

| FacultyID (PK) | name | email |
|---|---|---|
| 91 | Rohit | rohit@gmail.com |
| 92 | Somanna | Somanna@gmail.com |
| 93 | Rakshit | rakshit@gmail.com |
| 94 | Asad | asad@gmail.com |
| 95 | Shilpa | Shilpa@gmail.com |
| 96 | Megha | megha@gmail.com |

## Note:

- Here we have the relationship between Course and Faculty, in MySQL to create the relationship between two entities or tables, we have to use the foreign key.
- Hence here the cardinality ration is one to one, in foreign key column we can observe that one course is taken by one faculty only.

Tap Academy

# What is foreign key?

1. A foreign key is a key which is used to create the relationship between the tables.
2. It is represented by dotted lines or dashed lines.
3. It can contain duplicate values as well as null values (empty data).
4. We can have more than one foreign key in a table.

**2. Second, we are taking department and faulty and creating one to many relationship.**

**DEPARTMENT**

| Departmentcode (PK) | Departmentname |
|---|---|
| 201 | CS |
| 202 | ECE |
| 203 | Mechanical |

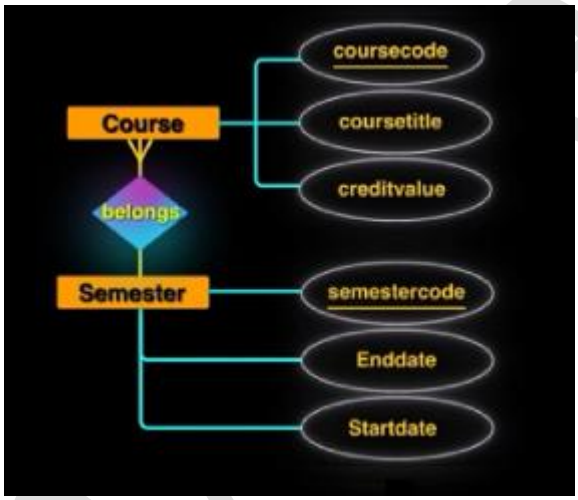
**FACULTY**

| FacultyID (PK) | name | email | Departmentcode (FK) |
|---|---|---|---|
| 91 | Rohit | rohit@gmail.com | 201 |
| 92 | Somanna | Somanna@gmail.com | 202 |
| 93 | Rakshit | rakshit@gmail.com | 203 |
| 94 | Asad | asad@gmail.com | 202 |
| 95 | Shilpa | Shilpa@gmail.com | 203 |
| 96 | Megha | megha@gmail.com | 201 |

## Note:

- Here the cardinality ratio between department and faculty is one to many and hence we can create the relationship by using the foreign key.
- Hence in foreign key column we can observe that in one department we have multiple faculties working.

### 3. Third we are taking course and semester entities and we will be creating the many to one relationship.




**SEMESTER**

| semestercode (PK) | Startdate | Enddate |
|---|---|---|
| 001 | 01/06/2002 | 25/12/2002 |
| 002 | 02/02/2003 | 30/07/2003 |
| 003 | 01/09/2003 | 30/03/2004 |
| 004 | 01/05/2004 | 30/11/2004 |

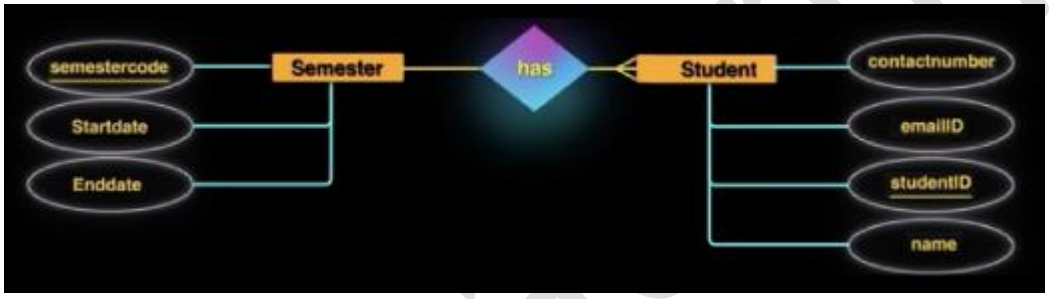
**COURSE**

| coursecode (PK) | coursetitle | creditvalue | semestercode (FK) |
|---|---|---|---|
| 101 | Dynamics | 4 | 001 |
| 102 | Fluid Mechanics | 3 | 002 |
| 103 | Java | 5 | 003 |
| 104 | C++ | 3 | 004 |
| 105 | Circuit Analysis | 2 | 002 |
| 106 | Digital electronics | 4 | 001 |
| 107 | Python | 2 | 003 |

## Note:

- Hence it is many to one cardinality ratio we can create the relationship by using the foreign key, here you can observe in course table, multiple courses are present in one semester.

### 4. Fourth we are taking semester and student entities and we will be creating one to many relationship between them.




**SEMESTER**

| semestercode (PK) | Startdate | Enddate |
|---|---|---|
| 001 | 01/06/2002 | 25/12/2002 |
| 002 | 02/02/2003 | 30/07/2003 |
| 003 | 01/09/2003 | 30/03/2004 |
| 004 | 01/05/2004 | 30/11/2004 |

**STUDENT**

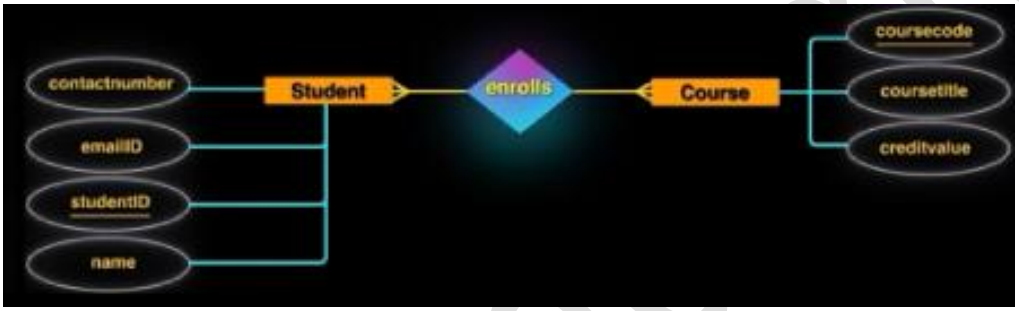
| studentID (PK) | name | email | contactnumber | semestercode (FK) |
|---|---|---|---|---|
| 301 | Rahul | rahul@gmail.com | 7816529144 | 001 |
| 302 | Krishna | krishna@gmail.com | 7895008587 | 002 |
| 303 | Sana | sana@gmail.com | 8073409794 | 003 |
| 304 | Savita | savita@gmail.com | 6363701421 | 004 |
| 305 | Satish | satish@gmail.com | 9620855028 | 001 |
| 306 | Divya | divya@gmail.com | 7338573928 | 002 |
| 307 | Akash | akash@gmail.com | 7022726043 | 003 |

TapAcademy

## Note:

- Hence it is one to many cardinality ratio we can create the relationship by using the foreign key, here you can observe in semester table, in one semester multiple student are present.

5. **Here we are taking student and course entities, and we will be creating many to many relationship between them.**




STUDENT

| studentID (PK) | name | email | contactnumber |
|---|---|---|---|
| 301 | Rahul | rahul@gmail.com | 7816529144 |
| 302 | Krishna | krishna@gmail.com | 7895008587 |
| 303 | Sana | sana@gmail.com | 8073409794 |
| 304 | Savita | savita@gmail.com | 6363701421 |
| 305 | Satish | satish@gmail.com | 9620855028 |
| 306 | Divya | divya@gmail.com | 7338573928 |
| 307 | Akash | akash@gmail.com | 7022726043 |

COURSE

| coursecode (PK) | coursetitle | creditvalue |
|---|---|---|
| 101 | Dynamics | 4 |
| 102 | Fluid Mechanics | 3 |
| 103 | Java | 5 |
| 104 | C++ | 3 |
| 105 | Circuit Analysis | 2 |
| 106 | Digital electronics | 4 |
| 107 | Python | 2 |

# Note:

- Here we have many to many relationship, so we cannot create the relationship between them by using the foreign key, if at all we try to create a relationship by using the foreign key, there can be a chance that we can lose a data.
- Hence here we need to go for alternate approach for creating the relationship, i.e., by using the composite primary key we can create the many to many relationship between any entities.
- To create a composite primary key, we have to take the key attribute from both the tables on which we need to create the relationship and that both the attributes together will be considered as one key.

**Composite Primary key**

| studentID | coursecode | coursetitle |
|---|---|---|
| 301 | 101 | Dynamics |
| 302 | 101 | Dynamics |
| 303 | 102 | Fluid Mechanics |
| 301 | 102 | Fluid Mechanics |
| 304 | 102 | Fluid Mechanics |
| 301 | 104 | C++ |
| 303 | 104 | C++ |

Tap Academy

## Note:

- Whenever we have the many to many relationships, we have to we have to make it as composite primary key.
- To create a relationship between the tables apart from taking the foreign key we can take the primary keys from two different table and together we can make it as a key which in technical called as composite primary key.
- In the above example, studentId is a primary key in student table and coursecode is a primary key in the course table, we took both the primary keys and made them as one composite primary key.
- In composite primary key the comparison will be done by taking both of the key values.
- Example:

| studentID | coursecode | coursetitle |
|---|---|---|
| 301 | 101 | Dynamics |
| 302 | 101 | Dynamics |
| 303 | 102 | Fluid Mechanics |
| 301 | 102 | Fluid Mechanics |
| 304 | 102 | Fluid Mechanics |
| 301 | 104 | C++ |
| 303 | 104 | C++ |

Tap Academy

Tap Academy

Tap Academy

# Data Types

## Numeric Data Types:

- **TINYINT:** Small integer, range from -128 to 127 or 0 to 255 (unsigned)
- **SMALLINT:** Integer, range from -32,768 to 32,767 or 0 to 65,535 (unsigned)
- **MEDIUMINT:** Integer, range from -8,388,608 to 8,388,607 or 0 to 16,777,215(unsigned)
- **INT or INTEGER:** Integer, range from -2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 (unsigned)
- **BIGINT**: Large integer, range from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 or 0 to 18,446,744,073,709,551,615 (unsigned)
- **DECIMAL or NUMERIC:** Exact fixed-point number with user-defined precision and scale
- **FLOAT:** Approximate floating-point number with single precision
- **DOUBLE:** Approximate floating-point number with double precision

## Character Data Types:

- **CHAR:** Fixed-length character string
- **VARCHAR:** Variable-length character string with a specified maximum length

## Binary Data Type:

- **BLOB:** Variable-length binary data with a maximum length of 65,535 bytes

## Date and Time Data Types:

- **DATE:** Date values, format 'YYYY-MM-DD'
- **TIME:** Time values, format 'hh:mm:ss'

TAP ACADEMY

SQL Datatypes Notes

- **DATETIME:** Date and time values, format 'YYYY-MM-DD hh:mm:ss'

TAP ACADEMY

- **TIMESTAMP:** Date and time values, format 'YYYY-MM-DD hh:mm:ss', automatically updates on record modification

- **YEAR:** Year values, format 'YYYY' or 'YY'

## Example 1:

```
create table student
(
s_id int,
name varchar(15),
age int,
gender char(1),
marks float,
dob date
);

insert into student values(1, "Arjun", 21, "M", 98.78, "2002-10-01");
insert into student values(2, "Raju", 23, "M", 95.78, "2000-10-01");
insert into student values(3, "Ravi", 20, "M", 90.78, "2003-10-01");
insert into student values(4, "Hari", 19, "M", 93.78, "2004-10-01");
insert into student values(5, "Rahul", 18, "M", 96.78, "2005-10-01");
```

TAP ACADEMY

select * from student;

| s_id | name | age | gender | marks | dob |
|---|---|---|---|---|---|
| 1 | Arjun | 21 | M | 98.78 | 2002-10-01 |
| 2 | Raju | 23 | M | 95.78 | 2000-10-01 |
| 3 | Ravi | 20 | M | 90.78 | 2003-10-01 |
| 4 | Hari | 19 | M | 93.78 | 2004-10-01 |
| 5 | Rahul | 18 | M | 96.78 | 2005-10-01 |

## Example 2:

```
CREATE TABLE Car (
    car_id INT PRIMARY KEY,
    brand VARCHAR(50),
    model VARCHAR(50),
    year YEAR,
    km_driven BIGINT,
    price DECIMAL(10, 2),
    engine_displacement FLOAT,
    fuel_efficiency DOUBLE,
    production_time DATETIME,
    last_updated TIMESTAMP,
    registered_at TIME,
    owners TINYINT,
    service_records SMALLINT,
    accidents MEDIUMINT
);
```

INSERT INTO Car (car_id, brand, model, year, km_driven, price, engine_displacement, fuel_efficiency, production_time, last_updated,

TAP ACADEMY

registered_at, owners, service_records, accidents)

TAP ACADEMY

VALUES

(1, 'Toyota', 'Corolla', 2022, 50000, 150000.00, 1.6, 30.5, '2021-03-10 14:30:00', '2023-03-10 15:30:00', '12:00:00', 2, 5, 0),

(2, 'Honda', 'Civic', 2021, 40000, 170000.50, 1.8, 28.7, '2021-03-10 14:30:00', '2024-03-10 18:30:00', '15:30:00', 1, 3, 1),

(3, 'Ford', 'Mustang', 2023, 15000, 300000.75, 2.3, 22.1, '2023-06-20 11:45:00', '2025-03-10 19:30:00', '09:45:00', 3, 8, 2),

(4, 'Chevrolet', 'Camaro', 2022, 200000, 28000.00, 2.5, 23.9, '2022-08-05 16:00:00', '2026-03-20 18:30:00', '17:30:00', 1, 2, 0),

(5, 'Nissan', 'Altima', 2022, 350000, 16500.00, 2.0, 29.4, '2022-04-18 10:15:00', '2027-03-10 13:30:00', '10:00:00', 2, 4, 1);

select * from car;

| car_id | brand | model | year | km_driven | price | engine_displacement | fuel_efficiency | production_time | last_updated | registered_at | owners | service_records | accidents |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Toyota | Corolla | 2022 | 50000 | 150000.00 | 1.6 | 30.5 | 2021-03-10 14:30:00 | 2023-03-10 15:30:00 | 12:00:00 | 2 | 5 | 0 |
| 2 | Honda | Civic | 2021 | 40000 | 170000.50 | 1.8 | 28.7 | 2021-03-10 14:30:00 | 2024-03-10 18:30:00 | 15:30:00 | 1 | 3 | 1 |
| 3 | Ford | Mustang | 2023 | 15000 | 300000.75 | 2.3 | 22.1 | 2023-06-20 11:45:00 | 2025-03-10 19:30:00 | 09:45:00 | 3 | 8 | 2 |
| 4 | Chevrolet | Camaro | 2022 | 200000 | 28000.00 | 2.5 | 23.9 | 2022-08-05 16:00:00 | 2026-03-20 18:30:00 | 17:30:00 | 1 | 2 | 0 |
| 5 | Nissan | Altima | 2022 | 350000 | 16500.00 | 2 | 29.4 | 2022-04-18 10:15:00 | 2027-03-10 13:30:00 | 10:00:00 | 2 | 4 | 1 |

## Difference between DATETIME and TIMESTAMP:

**DATETIME:** The DATETIME data type can store a range of dates from the year 1000 to the year 9999, along with corresponding time values.

**TIMESTAMP:** The TIMESTAMP data type can store dates from the year 1970 to the year 2038 (for a 32-bit system), with corresponding time values. The storage range can differ based on the system and the MySQL version used. On some systems, it can support a wider range beyond 2038.

**DATETIME:** DATETIME takes 8 bytes of storage.

**TIMESTAMP:** In most cases, TIMESTAMP takes 4 bytes of storage, which makes it more storage-efficient than DATETIME.

TAP ACADEMY

TAP ACADEMY

# Constraints

In SQL, constraints serve the purpose of enforcing rules and restrictions on the columns within tables. Various types of constraints are available for utilization in SQL, encompassing:

1. **NOT NULL:** This constraint ensures that a column cannot have a null value.

2. **UNIQUE:** This constraint ensures that a column or set of columns must contain unique values across all rows in the table.

3. **PRIMARY KEY:** This constraint is a combination of NOT NULL and UNIQUE constraints and is used to uniquely identify each row in a table.

4. **FOREIGN KEY:** This constraint ensures that a value in a column (or set of columns) matches the value in another table's primary key.

5. **CHECK:** This constraint ensures that a column's value satisfies a specific condition or range of values.

6. **DEFAULT:** This constraint sets a default value for a column if no value is specified during an INSERT operation.

Constraints are essential in maintaining data integrity and consistency in a database. They help ensure that the data entered into a table adheres to the rules and requirements of the database schema, preventing data inconsistencies and errors.

**Example 1:**

```
create table student
(
id int primary key,
name varchar(15) not null,
age int check(age>=18),
institute varchar(20) default "Tap Academy"
);
```

Tap Academy

insert into student values(1, "Arjun", 23, "Tap Academy");

insert into student values(2, "Sam", 19, null);

insert into student(id, name, age) values(3, "Suma", 19);

select * from student;

| id | name | age | institute |
|---|---|---|---|
| 1 | Arjun | 23 | Tap Academy |
| 2 | Sam | 19 | NULL |
| 3 | Suma | 19 | Tap Academy |

**Example 2:**

```
create table dept
(
d_id int primary key,
d_name varchar(15)
);

create table emp
(
id int primary key,
name varchar(20) not null,
age int,
d_id int,
foreign key(d_id) references dept(d_id)
);
```

Tap Academy

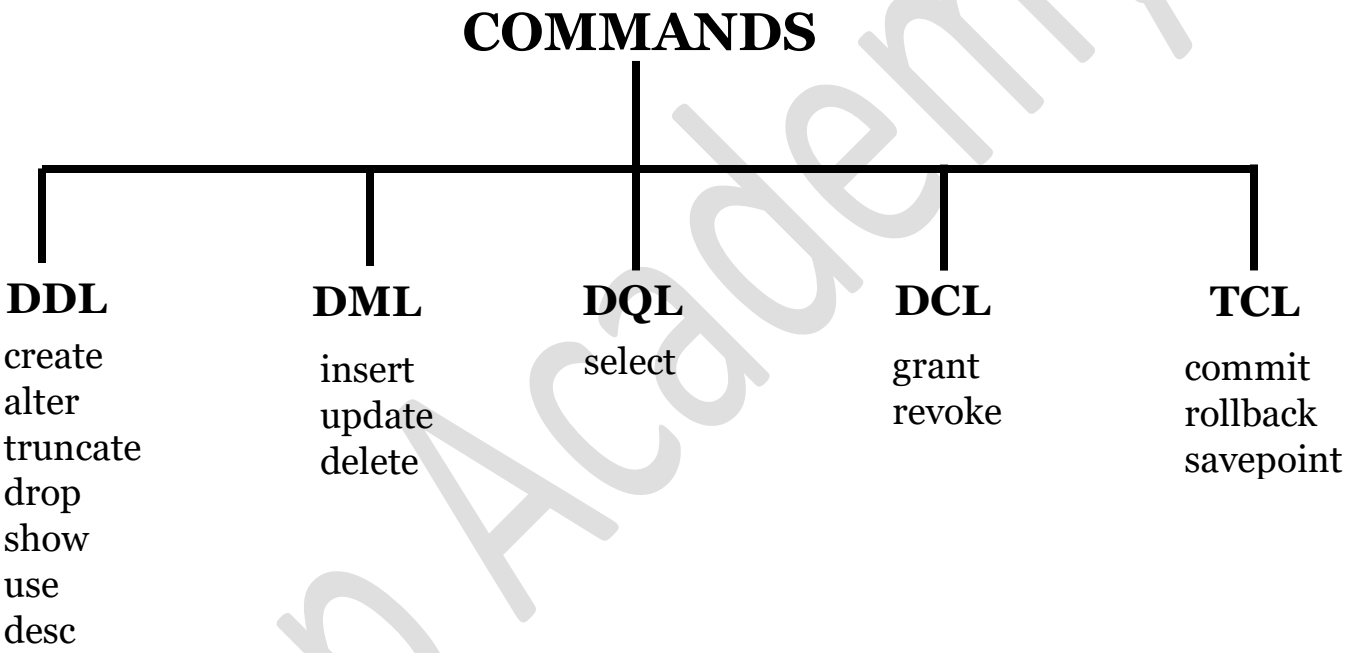
Tap Academy

# CRUD

Crud stands for create, read, update and delete.

To do any crud operators in MySQL we need use the commands.

We have different Types of commands/different types of SQL languages such as:




## Consider the below table:

```
mysql> desc student;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| id    | int         | YES  |     | NULL    |       |
| name  | varchar(10) | YES  |     | NULL    |       |
| marks | float       | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
```

Tap Academy

**Note:** On the above table all the below operations will be done.

## Adding a column into the existing student table

### Syntax:

Alter table table_name

Add Column_name datatype(size);

### Example:

alter table student

add gender char(1)

```
mysql> desc student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | YES  |     | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
| marks  | float       | YES  |     | NULL    |       |
| gender | char(1)     | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

Tap Academy

# Adding a column into the existing student table at the specified placed.

## Syntax:

Alter table table_name

Add Column_name datatype(size)

after Column_name;

## Example:

alter table student

add age int

after name;

```
mysql> desc student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | YES  |     | NULL    |       |
| name   | varchar(10) | YES  |     | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
| marks  | float       | YES  |     | NULL    |       |
| gender | char(1)     | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

## Renaming a column in the existing student table.

### Syntax:

alter table table_name

rename column old_column_name to new_column_name;

### Example:

alter table student

rename column name to fname;

```
mysql> desc student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | YES  |     | NULL    |       |
| fname  | varchar(10) | YES  |     | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
| marks  | float       | YES  |     | NULL    |       |
| gender | char(1)     | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

## Changing the Datatype of a column from the existing student table.

### Syntax

alter table table_name

modify column_name new_datatype;

## Example:

Alter table student

modify gender varchar(6);

```
mysql> desc student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | YES  |     | NULL    |       |
| fname  | varchar(10) | YES  |     | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
| marks  | float       | YES  |     | NULL    |       |
| gender | varchar(6)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

## Changing the size of a datatype from the existing student table.

## Syntax

alter table table_name

modify column_name datatype(new_size);

## Example:

Alter table student

modify fname varchar(20);

```
mysql> desc student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | YES  |     | NULL    |       |
| fname  | varchar(20) | YES  |     | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
| marks  | float       | YES  |     | NULL    |       |
| gender | varchar(6)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

## Removing a column from the existing table

### Syntax

alter table table_name

drop column_name;

### Example:

alter table student

drop marks;

```
mysql> desc student;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | YES  |     | NULL    |       |
| fname  | varchar(20) | YES  |     | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
| gender | varchar(6)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

## Renaming the existing table name

### Syntax:

alter table old_table_name

rename to new_table_name;

### Example:

alter table student

rename to student_details;

```
mysql> desc student_details;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| id     | int         | YES  |     | NULL    |       |
| fname  | varchar(20) | YES  |     | NULL    |       |
| age    | int         | YES  |     | NULL    |       |
| gender | varchar(6)  | YES  |     | NULL    |       |
+--------+-------------+------+-----+---------+-------+
```

## Inserting the records or data into the existing table:

### Type-1

### Syntax:

insert into table_name values (val1, val2,.....valn);

### Example:

insert into student_details values(1,"Arjun",22,"Male");

```
mysql> select * from student_details;
+------+-------+------+--------+
| id   | fname | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | Male   |
+------+-------+------+--------+
1 row in set (0.00 sec)
```

---

## Type-2

## Syntax:

insert into table_name values (val1, val2, null,.....valn);

## Example:

insert into student_details values(2,”Arun”,null,”Male”);

```
mysql> select * from student_details;
+------+-------+------+--------+
| id   | fname | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | Male   |
|    2 | Arun  | NULL | Male   |
+------+-------+------+--------+
2 rows in set (0.00 sec)
```

---

## Type-3

## Syntax:

insert into table_name(col1,col2,...coln) values (val1, val2,.....valn);

## Example:

insert into student_details(id, fname,age) values(3,”Ramya”,24);

Tap Academy

```
mysql> select * from student_details;
+------+-------+------+--------+
| id   | fname | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | Male   |
|    2 | Arun  | NULL | Male   |
|    3 | Ramya |   24 | NULL   |
+------+-------+------+--------+
3 rows in set (0.00 sec)
```

## Updating the cell data inside the existing table:

**Note:** Here from the student_details table we are updating the age for the student arun whose id is 2.

```
mysql> select * from student_details;
+------+-------+------+--------+
| id   | fname | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | Male   |
|    2 | Arun  | NULL | Male   |
|    3 | Ramya |   24 | NULL   |
+------+-------+------+--------+
3 rows in set (0.00 sec)
```

## Syntax:

update table_name set column_name=value

where condition;

## Example:

update student_details set age=23

where id=2;

```
mysql> select *
    -> from student_details;
+------+-------+------+--------+
| id   | fname | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | Male   |
|    2 | Arun  |   23 | Male   |
|    3 | Ramya |   24 | NULL   |
+------+-------+------+--------+
```

## Removing the cell data inside the existing table:

**Note:** Here from the student_details table we are removing the age of the student Ramya whose id is 3.

### Syntax:

update table_name set column_name=null

where condition;

### Example:

update student_details set age=null

where id=3;

Tap Academy

```
mysql> select * from student_details;
+------+-------+------+--------+
| id   | fname | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | Male   |
|    2 | Arun  |   23 | Male   |
|    3 | Ramya | NULL | NULL   |
+------+-------+------+--------+
```

## Removing all the records from the Existing table:

### Syntax:

truncate table table_name;

### Example:

truncate table student_details;

## Removing the entire existing table:

### Syntax:

drop table table_name;

### Example:

drop table student_details;

Tap Academy

## Removing the existing database:

### Syntax:

drop database database_name;

### Example:

drop database sqlnotes;

# DCL Commands

1. grant
2. revoke

**Grant:** Grant command is used to give the privilege to the user such as select, insert, update and delete.

**Note:** Before giving the privilege to the user you need to create the user first, and to create the user you need to be present in the root user, because in MySQL root username will be acting like an admin.

# Creating the user:

### Syntax:

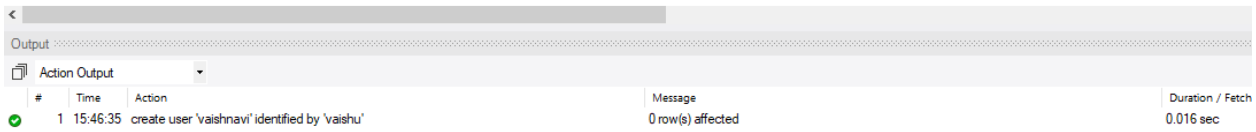
Create user 'user_name' identified by 'password'

### Example:

Create user 'vaishnavi' identified by 'vaishu'

Tap Academy

```
3
4
5 • create user 'vaishnavi' identified by 'vaishu';
```

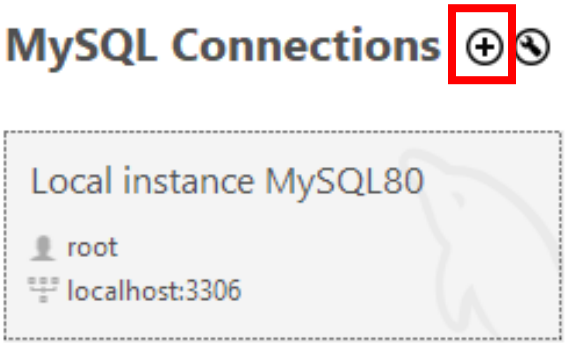

Output
Action Output
# Time Action Message Duration / Fetch
1 15:46:35 create user 'vaishnavi' identified by 'vaishu' 0 row(s) affected 0.016 sec

**Note:** After creating the user follow the below given steps.

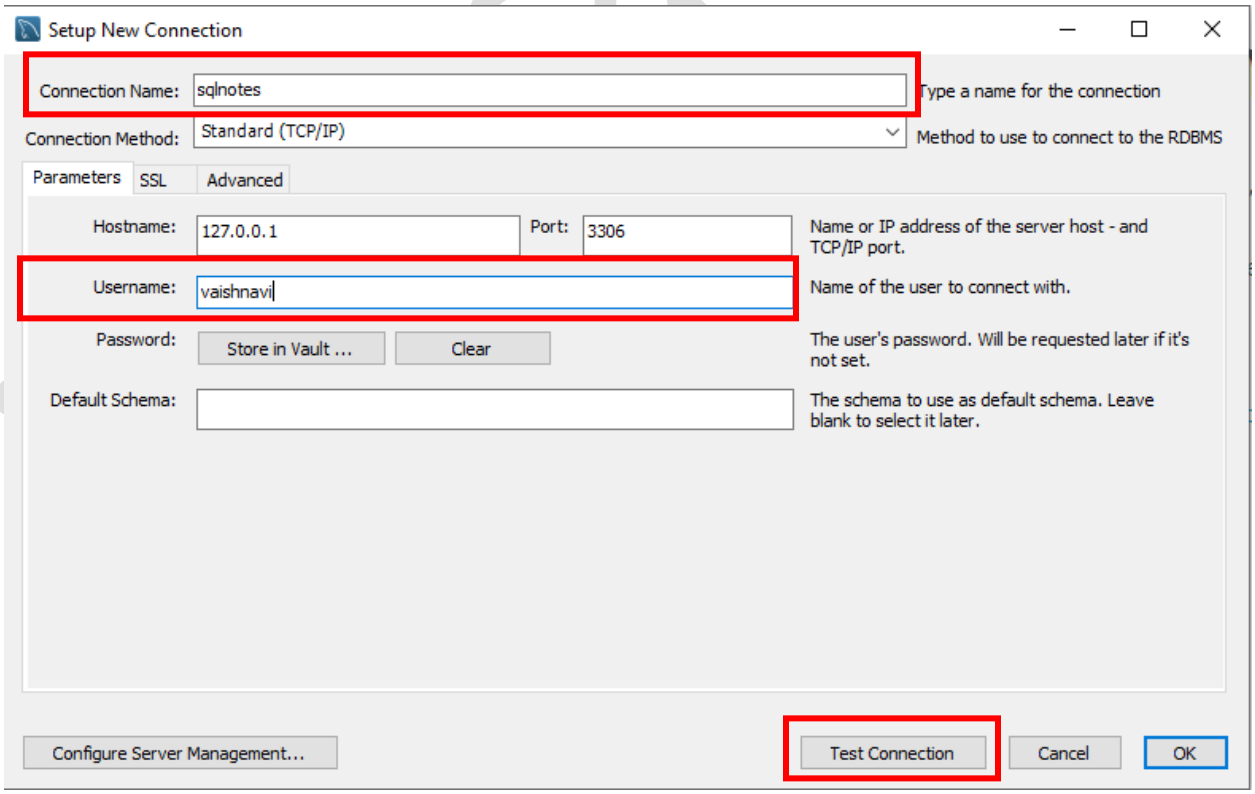
**Step1:** Go to the home page.

MySQL Workbench
Local instance MySQL80
File Edit View Database Tools Scripting Help
Welcome to MySQL Workbench
MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.
Browse Documentation >
Read the Blog >
Discuss on the Forums >
MySQL Connections
Local instance MySQL80
root
localhost:3306

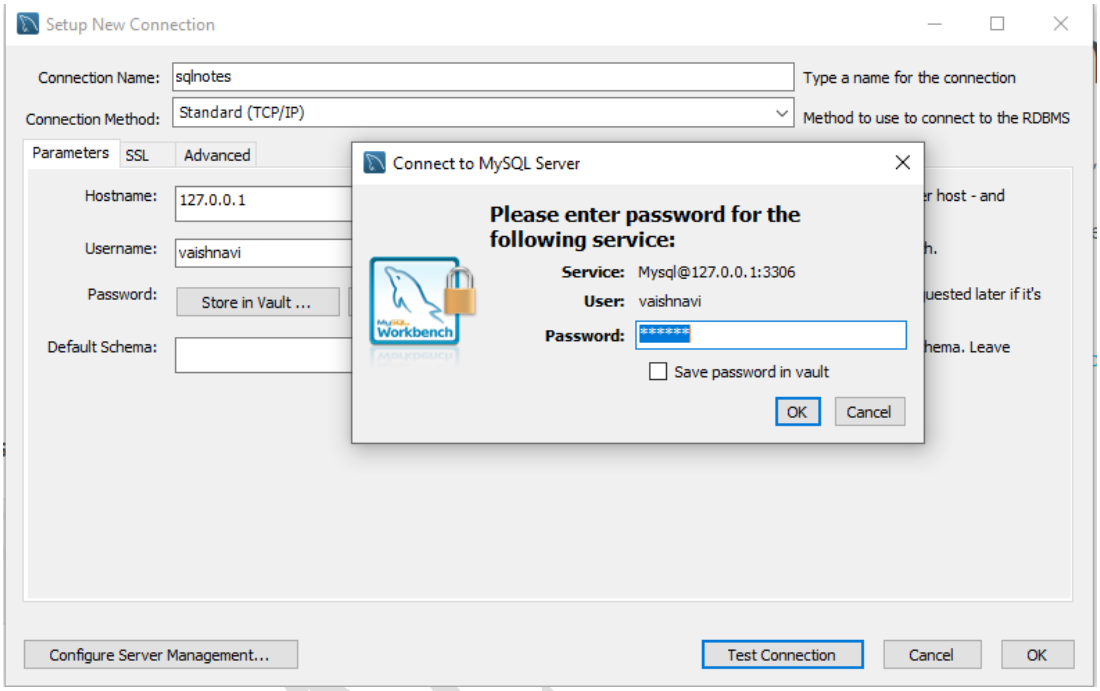
**Step2:** click on the (+) icon near the MySQL Connection.

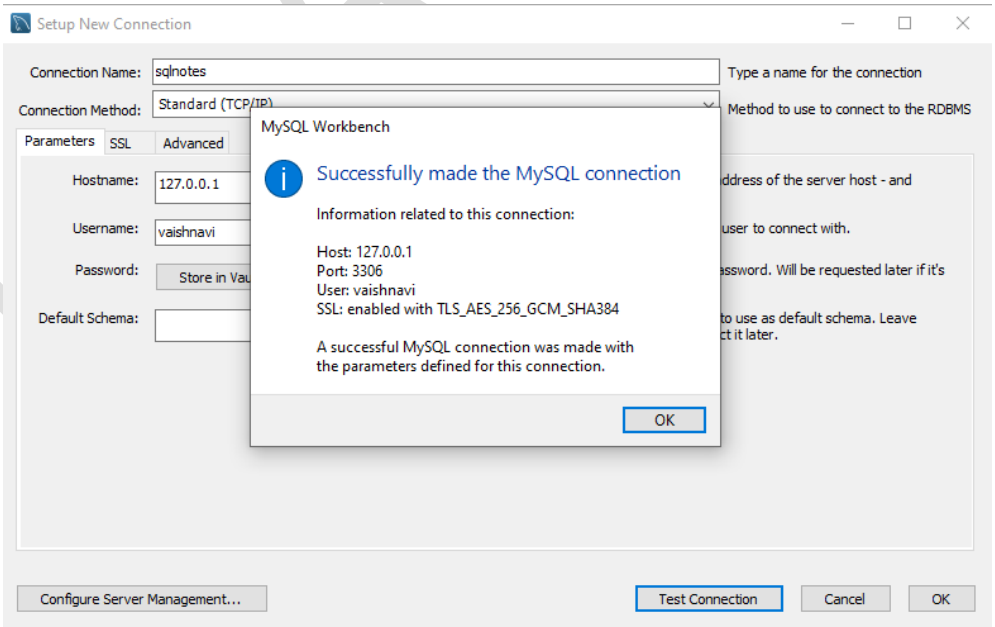



**Step3:** Give any name for the connection name, in the username give the name of the user which you have created and click on the Test connection button.

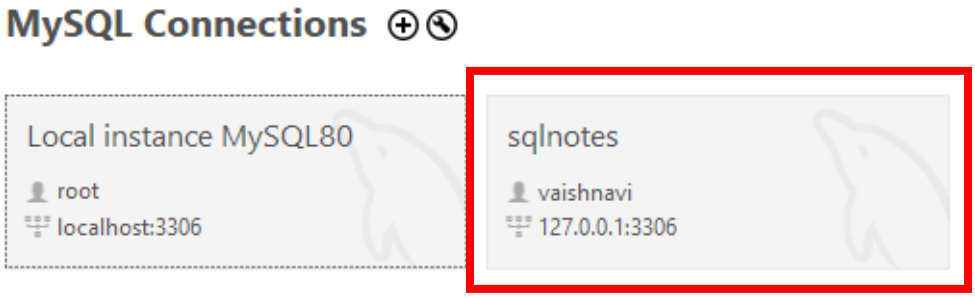


**Step4:** Give the password which you have given for the user while creating the user and click on ok button, then you will get one pop up as successfully made the MySQL connection, then just click on ok button.

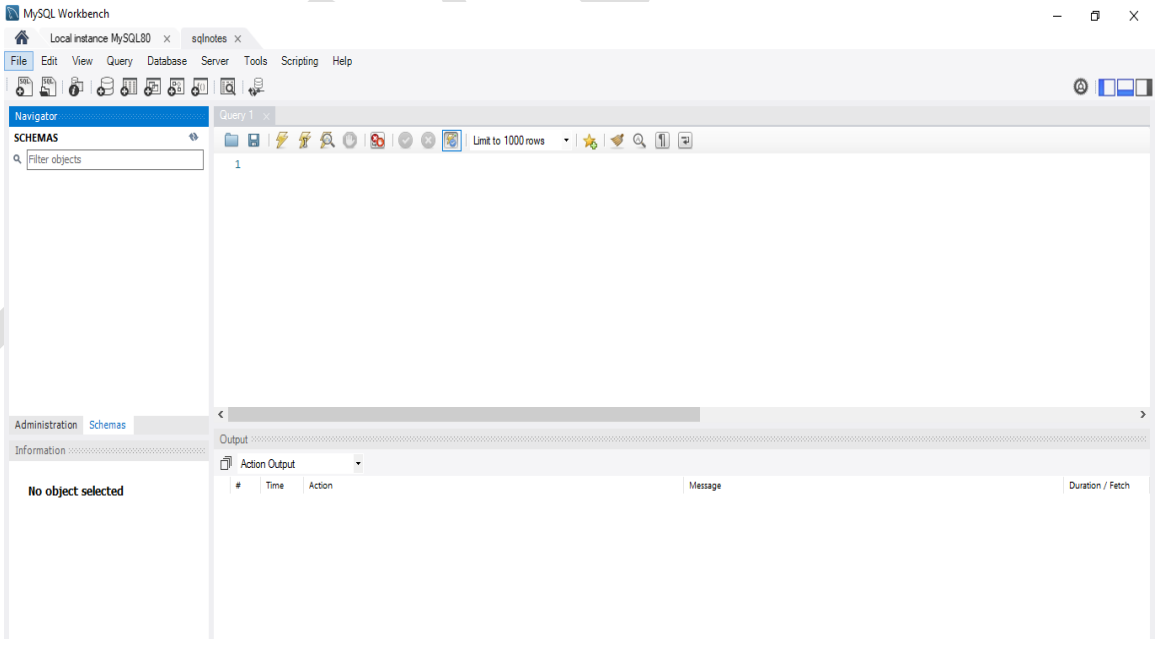


**Step5:** Go to the home then you can see the user which is created.




**Step6:** Click on the user which is sqlnotes you will be redirected to the page to perform the operations, but right now this user will not have any privilege hence she cannot perform any operations.



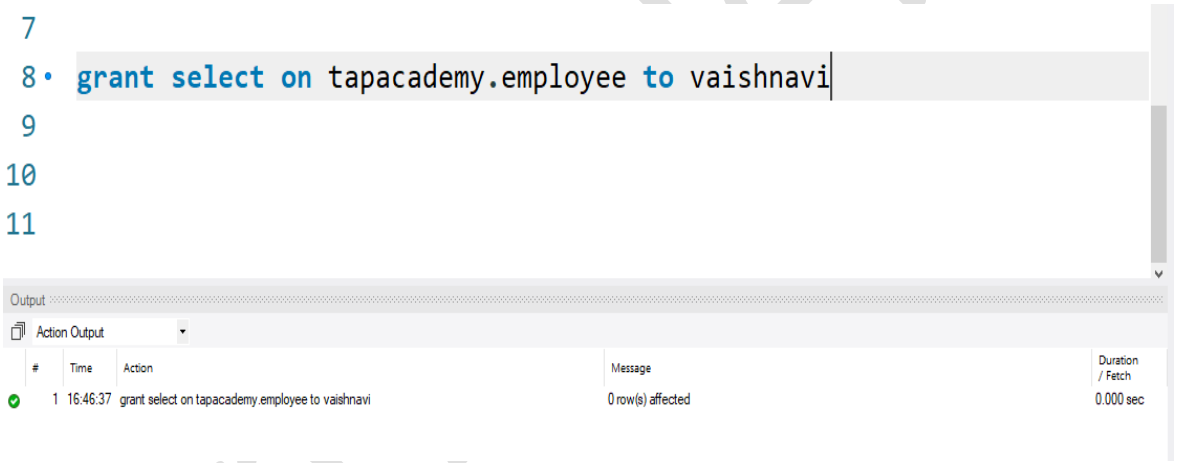
- **Step7:** If you want to give the privilege to the user go back to the root user and there give the privilege to the user by using the grant command.

## Syntax:

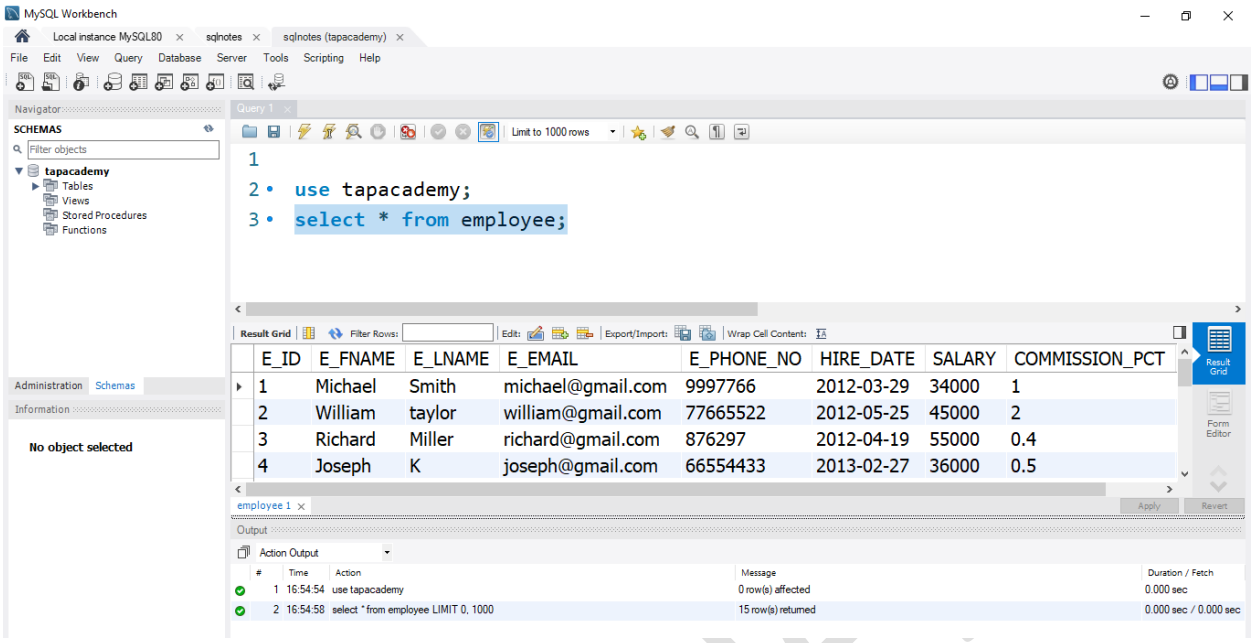
grant privilege on database_name.table_name to user_name

## Example:

grant select on tapacademy.employee to vaishnavi




**Note:** Here select is a privilege or permission which we are giving for the user vaishnavi. Now go back to the vaishnavi user, and refresh you will be able to see the tapacademy database and in that employee table you can select and display.




**Note:** In the vaishnavi user you can only select the data but cannot do any other operations because only select privilege has been given to that user.

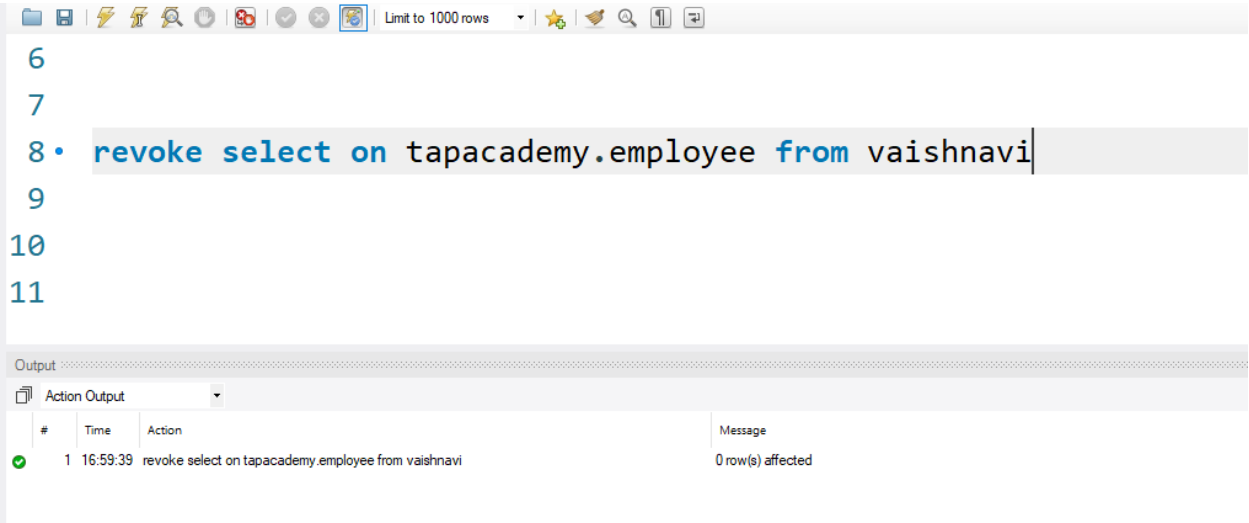
- **Step8:** If you want to take back the privilege from the user go back to the root user and there back the privilege from the user by using the revoke command.

## Syntax:

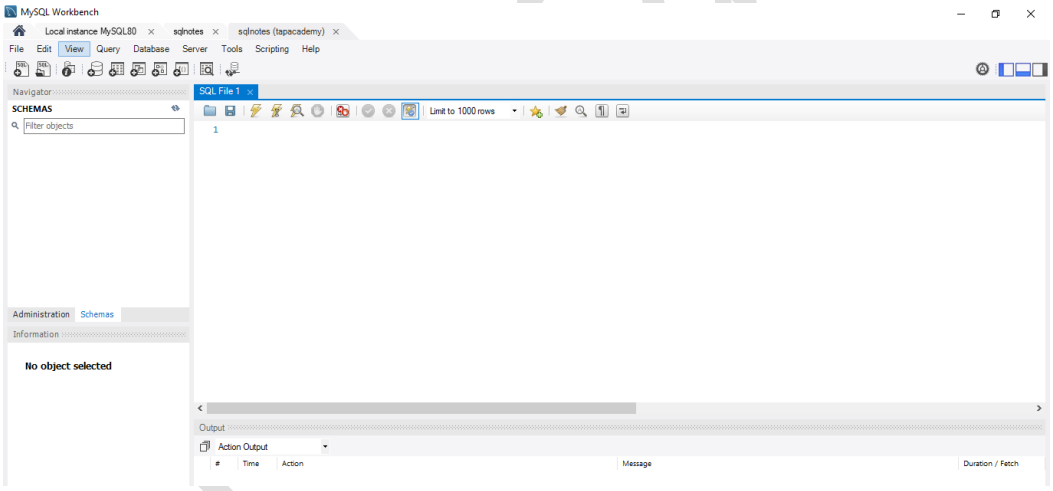
revoke privilege on database_name.table_name from user_name

## Example:

revoke select on tapacademy.employee from vaishnavi




**Note:** Once you take back the privilege if you go back to the vaishnavi user, you cannot select any data and database also will not be present, because the privilege has been removed.



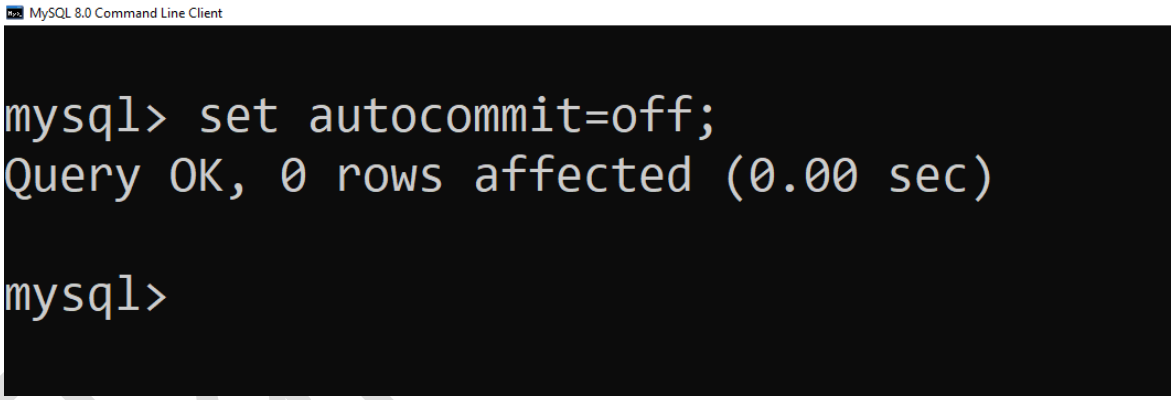

**Note:** This is how you can give privileges and take back the privileges from the user, not only select we can also give other privileges such as **insert**, **update** and **delete**, if we want to give all privileges then we can just give as **all,** the all privileges will be given to the user.

# TCL Commands

1. commit
2. rollback
3. savepoint

**commit:** commit command is used to save the data permanently in a system.

**Note:** In MySQL we no need of using the commit every time because, by default the commit will be on which will save the data permanently without using the commit command. If at all if you want to see the operations first you have to off the commit as shown below in the command prompt.



## Note:

- This TCL commands will only work on the DML operations.
- After turning off the commit, if you insert, update or delete any records into the table, it will not be saved permanently.
- If you want to save any records permanently the specifically you need to tell as commit.

```
mysql> set autocommit=off;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into student values(1,"Arjun",22,"M");
Query OK, 1 row affected (0.00 sec)

mysql> insert into student values(2,"Sam",22,"M");
Query OK, 1 row affected (0.00 sec)

mysql> insert into student values(3,"Paul",22,"M");
Query OK, 1 row affected (0.00 sec)

mysql> select * from student;
+------+-------+------+--------+
| id   | name  | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | M      |
|    2 | Sam   |   22 | M      |
|    3 | Paul  |   22 | M      |
+------+-------+------+--------+
3 rows in set (0.00 sec)
```

- The records which we have inserted into the student table it will not permanently saved as we turned off the commit, suppose if you close the command prompt and if you try to display the table it will give as empty set in command prompt as we did not give as commit.

```
mysql> select *
    -> from student;
Empty set (0.00 sec)
```

**rollback:** This command is used to rollback to the point, or go back to the place where we have done the commit lastly.

```
mysql> set autocommit=off;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into student values(1,"Arjun",22,"M");
Query OK, 1 row affected (0.00 sec)

mysql> insert into student values(2,"Sam",22,"M");
Query OK, 1 row affected (0.00 sec)

mysql> insert into student values(3,"Paul",22,"M");
Query OK, 1 row affected (0.00 sec)

mysql> select * from student;
+------+-------+------+--------+
| id   | name  | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | M      |
|    2 | Sam   |   22 | M      |
|    3 | Paul  |   22 | M      |
+------+-------+------+--------+
3 rows in set (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.01 sec)
```

In the above we have inserted 3 records and we have said as commit so all the 3 records will be saved permanently.

Now again we will insert 2 records here.

```
mysql> commit;
Query OK, 0 rows affected (0.01 sec)

mysql> insert into student values(4,"Pooja",23,"F");
Query OK, 1 row affected (0.00 sec)

mysql> insert into student values(5,"Roopa",23,"F");
Query OK, 1 row affected (0.00 sec)

mysql> select * from student;
+------+-------+------+--------+
| id   | name  | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | M      |
|    2 | Sam   |   22 | M      |
|    3 | Paul  |   22 | M      |
|    4 | Pooja |   23 | F      |
|    5 | Roopa |   23 | F      |
+------+-------+------+--------+
5 rows in set (0.00 sec)
```

In the above the last 2 records which we have inserted have not saved permanently because we did not tell commit for this records, now if we wanted to remove the last two records we have two options one is close the application and reopen it again, other one is use the delete command and remove the records specifically, but both is this approaches are not efficient, hence we can use the rollback command where it will go the place where we have done the commit lastly as show below:

```
mysql> select * from student;
+------+-------+------+--------+
| id   | name  | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | M      |
|    2 | Sam   |   22 | M      |
|    3 | Paul  |   22 | M      |
|    4 | Pooja |   23 | F      |
|    5 | Roopa |   23 | F      |
+------+-------+------+--------+
5 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from student;
+------+-------+------+--------+
| id   | name  | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | M      |
|    2 | Sam   |   22 | M      |
|    3 | Paul  |   22 | M      |
+------+-------+------+--------+
3 rows in set (0.00 sec)
```

**Savepoint:** By using this command we can create a point for every statement or for set of statements then we can rollback to the specific point instead of going back to place where we have done the commit lastly.

```
mysql> select * from student;
+------+-------+------+--------+
| id   | name  | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | M      |
|    2 | Sam   |   22 | M      |
|    3 | Paul  |   22 | M      |
+------+-------+------+--------+
3 rows in set (0.00 sec)

mysql> insert into student values(4,"Pooja",23,"F");
Query OK, 1 row affected (0.01 sec)

mysql> savepoint p1;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into student values(5,"Roopa",23,"F");
Query OK, 1 row affected (0.00 sec)

mysql> savepoint p2;
Query OK, 0 rows affected (0.00 sec)
```

In the above for the 4th record or statement we have created a point and for the 5th record also we have created the point. But both of the records are not still saved permanently because we did not tell here as commit, if we wanted to remove only the last record, we have to use only the delete command and delete it specifically, but instead of that we can rollback to the point where all the statements above that point will remain same, but

after that point all statements will be removed if we have not done the commit.

```
+------+-------+------+--------+
| id   | name  | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | M      |
|    2 | Sam   |   22 | M      |
|    3 | Paul  |   22 | M      |
|    4 | Pooja |   23 | F      |
|    5 | Roopa |   23 | F      |
+------+-------+------+--------+
5 rows in set (0.00 sec)

mysql> rollback to p1;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from student;
+------+-------+------+--------+
| id   | name  | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | M      |
|    2 | Sam   |   22 | M      |
|    3 | Paul  |   22 | M      |
|    4 | Pooja |   23 | F      |
+------+-------+------+--------+
```

Here you can see that only last record has been remove, we have only the records which was present above the p1 point. Still here the last record is not saved permanently because we did not tell it as commit, if you want to save it permanently then tell it as commit.

```
mysql> rollback to p1;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from student;
+------+-------+------+--------+
| id   | name  | age  | gender |
+------+-------+------+--------+
|    1 | Arjun |   22 | M      |
|    2 | Sam   |   22 | M      |
|    3 | Paul  |   22 | M      |
|    4 | Pooja |   23 | F      |
+------+-------+------+--------+


mysql> commit;
Query OK, 0 rows affected (0.01 sec)
```

Tap Academy

# Employee

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 5 | John | Williams | john@gmail.com | 7456465 | 2010-12-15 | 18000 | 1.5 | 25 |
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 8 | Aaron | Samuels | araon@gmail.com | 5423424 | 2014-06-24 | 12000 | 5.5 | 26 |
| 9 | Adam | Sam | adam@gmail.com | 325674 | 2015-04-16 | 53000 | 1.2 | 26 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |
| 14 | Savitha | Singh | savi@gmail.com | 3443333 | 2022-03-23 | 12000 | 0.15 | 23 |
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |

# Department

| D_ID | D_NAME |
|---|---|
| 20 | TRAINING |
| 21 | SALES |
| 22 | DIGITAL MARKETING |
| 23 | OPERATIONS |
| 24 | CREATIVE TEAM |
| 25 | BUssiness Dev |
| 26 | FINANCE |

# Product

| product_id | product_name | price | stock |
|---|---|---|---|
| 1 | Smartphone | 25000 | 100 |
| 2 | Laptop | 60000 | 50 |
| 3 | Tablet | 15000 | 75 |
| 4 | Headphones | 2000 | 200 |
| 5 | Smartwatch | 12000 | 40 |

# Operators

Operator is a reserved keyword which will be used to fetch the data efficiently according to our requirement and to perform a certain operation.

## Types of an operators:

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators

# Arithmetic Operators

Arithmetic operators are used to perform an arithmetic operation such as addition, subtraction, multiplication and division.

## Types of arithmetic operators:

1. **+ (addition):** It is used to perform the addition.
2. **- (subtraction):** It is used to perform the subtraction.
3. * **(multiplication):** It is used to perform the multiplication.
4. **/ (division):** It is used to perform the division.
5. **% (modulus):** It is used to get the remainder after performing the division.

**Note:** Arithmetic operators are the symbolic operators while writing a query we need to use only the symbols as +, - etc.,

Tap Academy

# Queries on Arithmetic Operators

## 1. Write a query to display employee id, employee name and salary by increasing the salary of 3000rs.

### Solution:

```
select e_id, e_fname, e_lname, salary+3000
from employee;
```

### Output:

| e_id | e_fname | e_lname | salary+3000 |
|---|---|---|---|
| 1 | Michael | Smith | 37000 |
| 2 | William | taylor | 48000 |
| 3 | Richard | Miller | 58000 |
| 4 | Joseph | K | 39000 |
| 5 | John | Williams | 21000 |
| 6 | Thomas | Raj | 88000 |
| 7 | Andy | Smith | 31500 |
| 8 | Aaron | Samuels | 15000 |
| 9 | Adam | Sam | 56000 |
| 10 | Nancy | A | 56000 |
| 11 | Madavan | Manish | 28000 |
| 12 | Jennifer | Anderson | 48500 |
| 13 | Mamatha | T | 49000 |
| 14 | Savitha | Singh | 15000 |
| 15 | Susan | Kumar | 22500 |

15 rows in set (0.00 sec)

**Note:** In the above query we are increasing the salary by 3000rs by using the + operator for all the employees in an employee table, but it will not permanently affect your original table, because we are using the select command so it will only fetch the data according to our requirement and we are using the + operator in the select statement because after performing the

operation it should be displayed as well, hence if any thing that should be present in the result then we need to give that in the select statement only.

## 2. Write a query to display employee id, employee name and salary by decreasing the salary of 1000rs.

### Solution:

```
select e_id, e_fname, e_lname, salary-1000
from employee;
```

### Output:

| e_id | e_fname | e_lname | salary-1000 |
|---|---|---|---|
| 1 | Michael | Smith | 33000 |
| 2 | William | taylor | 44000 |
| 3 | Richard | Miller | 54000 |
| 4 | Joseph | K | 35000 |
| 5 | John | Williams | 17000 |
| 6 | Thomas | Raj | 84000 |
| 7 | Andy | Smith | 27500 |
| 8 | Aaron | Samuels | 11000 |
| 9 | Adam | Sam | 52000 |
| 10 | Nancy | A | 52000 |
| 11 | Madavan | Manish | 24000 |
| 12 | Jennifer | Anderson | 44500 |
| 13 | Mamatha | T | 45000 |
| 14 | Savitha | Singh | 11000 |
| 15 | Susan | Kumar | 18500 |

## 3. Write a query to display employee id, employee name and salary by giving the hike of 10%.

### Solution:

```
select e_id, e_fname, e_lname, salary+(salary*10/100)
from employee;
```

### Output:

| e_id | e_fname | e_lname | salary+(salary*10/100) |
|---|---|---|---|
| 1 | Michael | Smith | 37400 |
| 2 | William | taylor | 49500 |
| 3 | Richard | Miller | 60500 |
| 4 | Joseph | K | 39600 |
| 5 | John | Williams | 19800 |
| 6 | Thomas | Raj | 93500 |
| 7 | Andy | Smith | 31350 |
| 8 | Aaron | Samuels | 13200 |
| 9 | Adam | Sam | 58300 |
| 10 | Nancy | A | 58300 |
| 11 | Madavan | Manish | 27500 |
| 12 | Jennifer | Anderson | 50050 |
| 13 | Mamatha | T | 50600 |
| 14 | Savitha | Singh | 13200 |
| 15 | Susan | Kumar | 21450 |

## 4. Write a query to display employee id, employee name and salary by reducing the hike of 20% whose employee id is 2.

### Solution:

```
select e_id, e_fname, e_lname, salary-(salary*20/100)
from employee
where e_id=2;
```

### Output:

| e_id | e_fname | e_lname | salary-(salary*20/100) |
|---|---|---|---|
| 2 | William | taylor | 36000 |

**Note:** Where clause is used to give the conditions which has to be satisfied by the records and it is used to check the conditions based on the requirement.

Tap Academy

## 5. Write a query to display the employee id, employee name and annual salary of each employee.

### Solution:

```
select e_id, e_fname, e_lname, salary*12
from employee;
```

### Output:

| e_id | e_fname | e_lname | salary*12 |
|---|---|---|---|
| 1 | Michael | Smith | 408000 |
| 2 | William | taylor | 540000 |
| 3 | Richard | Miller | 660000 |
| 4 | Joseph | K | 432000 |
| 5 | John | Williams | 216000 |
| 6 | Thomas | Raj | 1020000 |
| 7 | Andy | Smith | 342000 |
| 8 | Aaron | Samuels | 144000 |
| 9 | Adam | Sam | 636000 |
| 10 | Nancy | A | 636000 |
| 11 | Madavan | Manish | 300000 |
| 12 | Jennifer | Anderson | 546000 |
| 13 | Mamatha | T | 552000 |
| 14 | Savitha | Singh | 144000 |
| 15 | Susan | Kumar | 234000 |

**Note:** If we have to calculate the annual salary, we just need to multiple the salary by 12.

## 6. Write a query to display the employee id, employee name and half yearly salary of each employee.

### Solution:

```
select e_id, e_fname, e_lname, salary*6
from employee;
```

### Output:

| e_id | e_fname | e_lname | salary*6 |
|---|---|---|---|
| 1 | Michael | Smith | 204000 |
| 2 | William | taylor | 270000 |
| 3 | Richard | Miller | 330000 |
| 4 | Joseph | K | 216000 |
| 5 | John | Williams | 108000 |
| 6 | Thomas | Raj | 510000 |
| 7 | Andy | Smith | 171000 |
| 8 | Aaron | Samuels | 72000 |
| 9 | Adam | Sam | 318000 |
| 10 | Nancy | A | 318000 |
| 11 | Madavan | Manish | 150000 |
| 12 | Jennifer | Anderson | 273000 |
| 13 | Mamatha | T | 276000 |
| 14 | Savitha | Singh | 72000 |
| 15 | Susan | Kumar | 117000 |

**Note:** If we have to calculate the half yearly salary, we just need to multiple the salary by 6.

## 7. Write a query to display the employee id, employee name and find the remainder after dividing the salary and commission percentage of an employee.

### Solution:

```
select e_id, e_fname, e_lname, salary%commission_pct
from employee;
```

### Output:

| e_id | e_fname | e_lname | salary%commission_pct |
|---|---|---|---|
| 1 | Michael | Smith | 0 |
| 2 | William | taylor | 0 |
| 3 | Richard | Miller | 0.399180442094802 86 |
| 4 | Joseph | K | 0 |
| 5 | John | Williams | 0 |
| 6 | Thomas | Raj | 0.3966224193572998 |
| 7 | Andy | Smith | 0.600754976272583 |
| 8 | Aaron | Samuels | 4.5 |
| 9 | Adam | Sam | 0.7978940010070801 |
| 10 | Nancy | A | 0.7978940010070801 |
| 11 | Madavan | Manish | 0.09900659322738647 |
| 12 | Jennifer | Anderson | 0.19932200014591217 |
| 13 | Mamatha | T | 0.39817214012145996 |
| 14 | Savitha | Singh | 0.14952316880226135 |
| 15 | Susan | Kumar | 1.1992251873016357 |

## The execution flow of a query

**select----------3**

**from----------1**

**where----------2**

## Note

1. Select is a command which is used to select and display the data from the specified table.
2. From is a clause which is used to specify the table on which we need to fetch the values or perform any operation.
3. The commands and clauses in MySQL are not case sensitive.

Tap Academy

# Employee

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 5 | John | Williams | john@gmail.com | 7456465 | 2010-12-15 | 18000 | 1.5 | 25 |
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 8 | Aaron | Samuels | araon@gmail.com | 5423424 | 2014-06-24 | 12000 | 5.5 | 26 |
| 9 | Adam | Sam | adam@gmail.com | 325674 | 2015-04-16 | 53000 | 1.2 | 26 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |
| 14 | Savitha | Singh | savi@gmail.com | 3443333 | 2022-03-23 | 12000 | 0.15 | 23 |
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |

# Department

| D_ID | D_NAME |
|---|---|
| 20 | TRAINING |
| 21 | SALES |
| 22 | DIGITAL MARKETING |
| 23 | OPERATIONS |
| 24 | CREATIVE TEAM |
| 25 | BUssiness Dev |
| 26 | FINANCE |

# Product

| product_id | product_name | price | stock |
|---|---|---|---|
| 1 | Smartphone | 25000 | 100 |
| 2 | Laptop | 60000 | 50 |
| 3 | Tablet | 15000 | 75 |
| 4 | Headphones | 2000 | 200 |
| 5 | Smartwatch | 12000 | 40 |

# Comparison Operators

Comparison operators are used to compare the values or to check any conditions based on the requirement.

## Types of comparison operators:

1. **= (equal):** It is used to check if the specified values are equal.
2. **> (greater than):** It is used to check if the specified value is greater than the other specified value.
3. **< (less than):** It is used to check if the specified value is less than the other specified value.
4. **>= (greater than):** It is used to check if the specified value is greater than or equal to the other specified value.
5. **<= (less than):** It is used to check if the specified value is less than or equal to the other specified value.
6. **<> (Not equal):** It is used to check if the specified value is not equal to the other specified value.

**Note:** Comparison operators are the symbolic operators while writing a query we need to use only the symbols as =, > etc.,

Tap Academy

# Queries on Comparison Operators

## 1. Write a query to display all the products whose price is 15000.

### Solution:

```
select product_name
from product
where price=15000;
```

### Output:

```
+--------------+
| product_name |
+--------------+
| Tablet       |
+--------------+
```

**Note:** In the above query we are displaying the products, but the product price should be 15000, hence to compare if the price is equal to 15000, we need to use the =(Operator) in where clause to check the condition, in the similar way only we use other operators as well based on the requirement to check the condition or to compare the values on the specified values.

Tap Academy

## 2. Write a query to display employee id, employee name and salary whose salary is less than 20000rs.

### Solution:

```
select e_id, e_fname, e_lname, salary
from employee
where salary<20000;
```

### Output:

| e_id | e_fname | e_lname | salary |
|---|---|---|---|
| 5 | John | Williams | 18000 |
| 8 | Aaron | Samuels | 12000 |
| 14 | Savitha | Singh | 12000 |
| 15 | Susan | Kumar | 19500 |

Tap Academy

3. **Write a query to display employee id, employee name, commission percentage and salary whose commission percentage is greater than 1.**

## Solution:

```
select e_id, e_fname, e_lname, salary, commission_pct
from employee
where commission_pct>1;
```

## Output:

| e_id | e_fname | e_lname | salary | commission_pct |
|---|---|---|---|---|
| 2 | William | taylor | 45000 | 2 |
| 5 | John | Williams | 18000 | 1.5 |
| 8 | Aaron | Samuels | 12000 | 5.5 |
| 9 | Adam | Sam | 53000 | 1.2 |
| 10 | Nancy | A | 53000 | 1.2 |
| 13 | Mamatha | T | 46000 | 1.2 |
| 15 | Susan | Kumar | 19500 | 1.2 |

Tap Academy

## 4. Write a query to display all products whose price is greater than or equal to 25000.

### Solution:

```
select product_name
from product
where price>=25000;
```

### Output:

```
+--------------+
| product_name |
+--------------+
| Smartphone   |
| Laptop       |
+--------------+
```

Tap Academy

## 5. Write a query to display all products whose price is less than or equal to 25000.

### Solution:

```
select product_name
from product
where price<=25000;
```

### Output:

```
+--------------+
| product_name |
+--------------+
| Smartphone   |
| Tablet       |
| Headphones   |
| Smartwatch   |
+--------------+
```

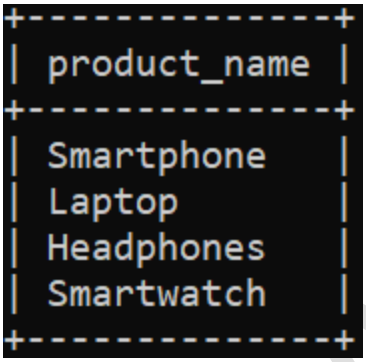
Tap Academy

## 6. Write a query to display all the products whose price is not 15000.

### Solution:

```
select product_name
from product
where price<>15000;
```

### Output:



| product_name |
|---|
| Smartphone |
| Laptop |
| Headphones |
| Smartwatch |

Tap Academy

## 7. Write a query to display details of employee whose employee id is either 14 or 15 but salary should be greater than or equal to 15000rs.

### Solution:

```
select *
from employee
where (e_id=14 or e_id=15) and salary>=15000;
```

### Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |

**Note:** In the above first or conditions should perform then only and condition it should check, so we need to priorities it by give in the parenthesis, then first or condition it will check then only and it will check.

Tap Academy

Tap Academy

# Employee

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 5 | John | Williams | john@gmail.com | 7456465 | 2010-12-15 | 18000 | 1.5 | 25 |
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 8 | Aaron | Samuels | araon@gmail.com | 5423424 | 2014-06-24 | 12000 | 5.5 | 26 |
| 9 | Adam | Sam | adam@gmail.com | 325674 | 2015-04-16 | 53000 | 1.2 | 26 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |
| 14 | Savitha | Singh | savi@gmail.com | 3443333 | 2022-03-23 | 12000 | 0.15 | 23 |
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |

# Department

| D_ID | D_NAME |
|---|---|
| 20 | TRAINING |
| 21 | SALES |
| 22 | DIGITAL MARKETING |
| 23 | OPERATIONS |
| 24 | CREATIVE TEAM |
| 25 | BUssiness Dev |
| 26 | FINANCE |

# Product

| product_id | product_name | price | stock |
|---|---|---|---|
| 1 | Smartphone | 25000 | 100 |
| 2 | Laptop | 60000 | 50 |
| 3 | Tablet | 15000 | 75 |
| 4 | Headphones | 2000 | 200 |
| 5 | Smartwatch | 12000 | 40 |

# Logical Operators

Logical operators are used to perform a logical operation and used to check the conditions based on the requirement.

## Types of logical operators:

1. **AND:** Whenever we want to give more than one condition and if all the given conditions should be satisfied then we need to use AND logical operator.
2. **OR:** Whenever we want to give more than one condition and if either of the given conditions should be satisfied then we need to use OR logical operator.
3. **DISTINCT:** It is used whenever we want to fetch the unique values from the specified columns.
4. **BETWEEN AND:** Whenever we want to check the condition in the range of values then we need to use BETWEEN AND logical operator.
   **Example**: Display the details of employee whose employee id is from 10 to 15.
5. **NOT BETWEEN AND:** Whenever we want to check the condition in the range of values and if that range should be excluded from the result, then we need to use NOT BETWEEN AND logical operator.
   **Example**: Display the details of employee whose employee id is not from 10 to 15.
6. **IN:** Whenever we want to check the condition with the set of values then we need to use IN logical operator.
   **Example**: Display the details of employee whose employee id is 9, 7, 10, or 15.
7. **NOT IN:** Whenever we want to check the condition with the set of values and if that set of values should be excluded from the result, then we need to use NOT IN logical operator.
   **Example**: Display the details of employee whose employee id is not 9, 7, 10, or 15.

Tap Academy

8. **IS NULL:** Whenever we want to check the absence of the data in the specified column then we need to use the IS NULL operator.
   **Example**: Display the details of employee whose do not take any commission.
9. **IS NOT NULL:** Whenever we want to check the presence of the data in the specified column then we need to use the IS NOT NULL operator.
   **Example**: Display the details of employee whose take commission.
10. **LIKE:** It is used for the pattern matching. Whenever they provide a partial information in the questions, we can take that pattern and we can match with the data present inside the table by using the LIKE operator. It has two special characters or wildcards such as:
    - **%(Modulus):** It will match zero (0) or more number of characters.
    - **_(Underscore):** It will match exactly one character.

Tap Academy

# Queries on Logical Operators

1. **Write a query to display all the products whose price is greater than 15000 and whose stock is 50 and above.**

## Solution:

```
select product_name
from product
where price>15000 and stock>=50;
```

## Output:

```
+--------------+
| product_name |
+--------------+
| Smartphone   |
| Laptop       |
+--------------+
```

Tap Academy

## 2. Write a query to display details of employee who works in the department 22 or 23.

### Solution:

```
select *
from employee
where d_id=22 or d_id=23;
```

### Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 14 | Savitha | Singh | savi@gmail.com | 3443333 | 2022-03-23 | 12000 | 0.15 | 23 |

Tap Academy

## 3. Write a query to display details of employee whose id is 12 and salary less than 25000rs.

### Solution:

```
select *
from employee
where e_id=12 and salary<25000;
```

### Output:

```
mysql> select *
    -> from employee
    -> where e_id=12 and salary<25000;
Empty set (0.00 sec)
```

**Note:** In the above query since we are giving AND logical operators both the condition should be satisfied if not it will not return any result as show in the above result.

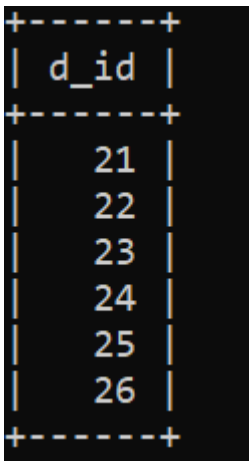
Tap Academy

## 4. Write a query to display the unique department id from the employee table.

### Solution:

```
select distinct d_id
from employee;
```

### Output:



```
+------+
| d_id |
+------+
|   21 |
|   22 |
|   23 |
|   24 |
|   25 |
|   26 |
+------+
```

Tap Academy

## 5. Write a query to display details of employee whose salary is in range from 20000 to 30000rs.

### Solution:

```
select *
from employee
where salary between 20000 and 30000;
```

### Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |

Tap Academy

## 6. Write a query to display details of employee whose salary is not in range from 10000 to 20000rs.

### Solution:

```
select *
from employee
where salary not between 10000 and 20000;
```

### Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 9 | Adam | Sam | adam@gmail.com | 325674 | 2015-04-16 | 53000 | 1.2 | 26 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |

Tap Academy

## 7. Write a query to display the details of department whose department id is 22,24 or 26.

### Solution:

```
select *
from department
where d_id IN(22,24,26);
```

### Output:

```
+------+------------------+
| D_ID | D_NAME           |
+------+------------------+
|   22 | DIGITAL MARKETING |
|   24 | CREATIVE TEAM    |
|   26 | FINANCE          |
+------+------------------+
3 rows in set (0.00 sec)
```

Tap Academy

## 8. Write a query to display the details of employee whose salary is not 18000, 12000,19000.

### Solution:

```
select *
from employee
where salary NOT IN(18000,12000,19000);
```

### Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 9 | Adam | Sam | adam@gmail.com | 325674 | 2015-04-16 | 53000 | 1.2 | 26 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |

Tap Academy

## 9. Write a query to display the details of employee who have an email id.

### Solution:

```
select *
from employee
where e_email IS NOT NULL;
```

### Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 5 | John | Williams | john@gmail.com | 7456465 | 2010-12-15 | 18000 | 1.5 | 25 |
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 8 | Aaron | Samuels | araon@gmail.com | 5423424 | 2014-06-24 | 12000 | 5.5 | 26 |
| 9 | Adam | Sam | adam@gmail.com | 325674 | 2015-04-16 | 53000 | 1.2 | 26 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |
| 14 | Savitha | Singh | savi@gmail.com | 3443333 | 2022-03-23 | 12000 | 0.15 | 23 |
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |

## 10. Write a query to display the details of employee who do not get any commission.

### Solution:

```
select *
from employee
where commission_pct IS NULL;
```

### Output:

```
mysql> select *
    -> from employee
    -> where commission_pct IS NULL;
Empty set (0.00 sec)
```

**Note:** In the above query it is not returning any result because in the employee table every employee is getting the commission hence its not returning any result.

Tap Academy

## 11. Write a query to display the details of employee whose first name starts with “W”.

### Solution:

```
select *
from employee
where e_fname like “W%”;
```

### Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |

**Note:** Whenever we want to match a pattern, we need to use the like operator and the pattern should be given inside the single or double quotes only. In the above query we are fetching the first names which are starting with “W” so we are giving as “W” first but after the first character we might have zero (0) or more number of characters hence we need to match all those characters/digits after the “W” character, so we need to give here as %(modulus) as it will match zero (0) or more number of characters after “W”.

TapAcademy

## 12. Write a query to display the details of employee whose last name ends with “a”.

### Solution:

```
select *
from employee
where e_lname like “%a”;
```

### Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |

Tap Academy

## 13. Write a query to display the details of department whose department name contains “in”.

### Solution:

```
select *
from department
where d_name like “%in%”;
```

### Output:

```
+------+------------------+
| D_ID | D_NAME           |
+------+------------------+
|   20 | TRAINING         |
|   22 | DIGITAL MARKETING |
|   25 | BUssiness Dev    |
|   26 | FINANCE          |
+------+------------------+
4 rows in set (0.01 sec)
```

Tap Academy

## 14. Write a query to display the details of department whose department name 3rd character is “G”.

### Solution:

```
select *
from department
where d_name like “__G%”;
```

### Output:

```
+------+--------------------+
| D_ID | D_NAME             |
+------+--------------------+
|   22 | DIGITAL MARKETING  |
+------+--------------------+
1 row in set (0.00 sec)
```

**Note:** Here one underscore will match exactly one character, in the above query 3rd character should be “G” so before “G” we need to have exactly two characters only, so we have given two underscores where each underscore will match one one character and 3rd character, we have given as “G” and after that we can have zero (0) or more number of characters so to match that we have given as %(modulus).

Tap Academy

## 15. Write a query to display the details of department whose department name last 2nd character is “E”.

### Solution:

```
select *
from department
where d_name like “%E_”;
```

### Output:

```
+------+---------------+
| D_ID | D_NAME        |
+------+---------------+
|   21 | SALES         |
|   25 | BUssiness Dev |
+------+---------------+
2 rows in set (0.00 sec)
```

**Note:** Whenever you’re giving an underscore make sure spaces should not be given in between them.

Tap Academy

## 16. Write a query to display the details of department whose department name contains exactly 7 characters.

### Solution:

```
select *
from department
where d_name like "_______";
```

### Output:

```
+------+---------+
| D_ID | D_NAME  |
+------+---------+
|   26 | FINANCE |
+------+---------+
1 row in set (0.00 sec)
```

Tap Academy

## 17. Write a query to display the details of employee whose first name do not start with “a”.

### Solution:

```
select *
from employee
where e_fname not like “a%”;
```

### Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 5 | John | Williams | john@gmail.com | 7456465 | 2010-12-15 | 18000 | 1.5 | 25 |
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |
| 14 | Savitha | Singh | savi@gmail.com | 3443333 | 2022-03-23 | 12000 | 0.15 | 23 |
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |

12 rows in set (0.00 sec)

**Note:** Whenever we want to exclude the pattern we can just use not like operators and then give the pattern.

Tap Academy

# Escape Character

- Whenever we want to match the %(modulus) and _(underscore) we cannot directly give that inside the pattern as % and _ are the special characters in the like operator.
- First, we have to convert that special characters as a normal character then only we can match them with the data present inside the table.
- To convert these special characters to normal characters we can use a “Escape Character”.
- This escape character can be anything such as “/”, ”@”, ”#” etc.,
- Consider the below table to solve the further queries using escape character.

```
mysql> select * from student;
+------+--------+------+--------+
| id   | name   | age  | gender |
+------+--------+------+--------+
|    1 | Arjun  |   22 | M      |
|    2 | S%am   |   22 | M      |
|    3 | Pa_ul  |   22 | M      |
|    4 | Poo%ja |   23 | F      |
+------+--------+------+--------+
4 rows in set (0.00 sec)
```

## 18. Write a query to display the details of student whose name contains %(modulus).

### Solution:

```
select *
from student
where name like "%/%%" escape "/";
```

### Output:

```
+------+--------+------+--------+
| id   | name   | age  | gender |
+------+--------+------+--------+
|    2 | S%am   |   22 | M      |
|    4 | Poo%ja |   23 | F      |
+------+--------+------+--------+
2 rows in set (0.00 sec)
```

**Note:** Here escape character "/" will convert the % special character as a normal character, and one escape character will convert only one special character at a time, and before that special character we need to give this escape character.

Tap Academy

## 19. Write a query to display the details of student whose name contains _(underscore).

### Solution:

```
select *
from student
where name like "%/_%" escape "/";
```

### Output:

```
+------+-------+------+--------+
| id   | name  | age  | gender |
+------+-------+------+--------+
|    3 | Pa_ul |   22 | M      |
+------+-------+------+--------+
1 row in set (0.00 sec)
```

Tap Academy

# Regular Expression

- It is also used for the pattern matching like the like operator.
- It will provide more efficient way of match the patterns.
- While writing the query instead of like operator we need to include regexp.

| Special Character | Functionality |
|---|---|
| ^(caret) | Matches the start of a character |
| $(dollar) | Matches the end of a character |
| .(dot operator) | Matches any single character |
| [abc] | Matches any data that contains either a or b or c |
| ^[abc] | Matches any data that start with a or b or c |
| [abc]$ | Matches any data that end with a or b or c |
| ^[^a\|b\|c] | Matches any data that not starting with a or b or c |
| [^a\|b\|c]$ | Matches any data that not ending with a or b or c |
| ^[a-d] | Matches any data that starts in the range from a to d |
| [a-d]$ | Matches any data that ends in the range from a to d |

Tap Academy

1. **Write a query to display the details of employee whose first name starts with "W".**

## Solution:

```
select *
from employee
where e_fname regexp "^W";
```

## Output:

```
+------+---------+---------+-------------------+------------+------------+--------+----------------+------+
| E_ID | E_FNAME | E_LNAME | E_EMAIL           | E_PHONE_NO | HIRE_DATE  | SALARY | COMMISSION_PCT | D_ID |
+------+---------+---------+-------------------+------------+------------+--------+----------------+------+
|    2 | William | taylor  | william@gmail.com |   77665522 | 2012-05-25 |  45000 |              2 |   23 |
+------+---------+---------+-------------------+------------+------------+--------+----------------+------+
1 row in set (0.00 sec)
```

2. **Write a query to display the details of employee whose last name ends with "a".**

## Solution:

```
select *
from employee
where e_lname regexp "a$";
```

## Output:

```
+------+---------+---------+-----------------+------------+------------+--------+----------------+------+
| E_ID | E_FNAME | E_LNAME | E_EMAIL         | E_PHONE_NO | HIRE_DATE  | SALARY | COMMISSION_PCT | D_ID |
+------+---------+---------+-----------------+------------+------------+--------+----------------+------+
|   10 | Nancy   | A       | nancy@gmail.com |    3453424 | 2016-10-25 |  53000 |            1.2 |   22 |
+------+---------+---------+-----------------+------------+------------+--------+----------------+------+
1 row in set (0.00 sec)
```

Tap Academy

3. **Write a query to display the details of department whose department name contains “in”.**

## Solution:

```
select *
from department
where d_name regexp “in”;
```

## Output:

```
+------+------------------+
| D_ID | D_NAME           |
+------+------------------+
|   20 | TRAINING         |
|   22 | DIGITAL MARKETING |
|   25 | BUssiness Dev    |
|   26 | FINANCE          |
+------+------------------+
4 rows in set (0.00 sec)
```

4. **Write a query to display the details of department whose department name 3rd character is “G”.**

## Solution:

```
select *
from department
where d_name regexp “^..G”;
```

## Output:

```
+------+------------------+
| D_ID | D_NAME           |
+------+------------------+
|   22 | DIGITAL MARKETING |
+------+------------------+
1 row in set (0.00 sec)
```

Tap Academy

5. **Write a query to display the details of department whose department name last 2nd character is "E".**

## Solution:

```
select *
from department
where d_name regexp "E.$";
```

## Output:

```
+------+--------------+
| D_ID | D_NAME       |
+------+--------------+
|   21 | SALES        |
|   25 | BUssiness Dev |
+------+--------------+
2 rows in set (0.00 sec)
```

6. **Write a query to display the details of department whose department name contains exactly 7 characters.**

## Solution:

```
select *
from department
where d_name regexp "^.......$";
```

## Output:

```
+------+---------+
| D_ID | D_NAME  |
+------+---------+
|   26 | FINANCE |
+------+---------+
```

Tap Academy

## 7. Write a query to display the details of department whose department name starts with vowels.

### Solution:

```
select *
from department
where d_name regexp ”^[aeiou]”;
```

### Output:

```
+------+------------+
| D_ID | D_NAME     |
+------+------------+
|   23 | OPERATIONS |
+------+------------+
1 row in set (0.00 sec)
```

## 8. Write a query to display the details of department whose department name starts with consonants.

### Solution:

```
select *
from department
where d_name not regexp ”^[aeiou]”;
```

### Output:

```
+------+------------------+
| D_ID | D_NAME           |
+------+------------------+
|   20 | TRAINING         |
|   21 | SALES            |
|   22 | DIGITAL MARKETING |
|   24 | CREATIVE TEAM    |
|   25 | BUssiness Dev    |
|   26 | FINANCE          |
+------+------------------+
6 rows in set (0.00 sec)
```

Tap Academy

9. **Write a query to display the details of department whose department name contains vowels.**

## Solution:

```
select *
from department
where d_name regexp ”[aeiou]”;
```

## Output:

```
+------+------------------+
| D_ID | D_NAME           |
+------+------------------+
|   20 | TRAINING         |
|   21 | SALES            |
|   22 | DIGITAL MARKETING |
|   23 | OPERATIONS       |
|   24 | CREATIVE TEAM    |
|   25 | BUssiness Dev    |
|   26 | FINANCE          |
+------+------------------+
7 rows in set (0.00 sec)
```

10. **Write a query to display the details of department whose department name contains consonants.**

## Solution:

```
select *
from department
where d_name not regexp ”[aeiou]”;
```

## Output:

```
mysql> select *
    -> from department
    -> where d_name not regexp "[aeiou]";
Empty set (0.00 sec)
```

Tap Academy

Tap Academy

# Employee

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 5 | John | Williams | john@gmail.com | 7456465 | 2010-12-15 | 18000 | 1.5 | 25 |
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 8 | Aaron | Samuels | araon@gmail.com | 5423424 | 2014-06-24 | 12000 | 5.5 | 26 |
| 9 | Adam | Sam | adam@gmail.com | 325674 | 2015-04-16 | 53000 | 1.2 | 26 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |
| 14 | Savitha | Singh | savi@gmail.com | 3443333 | 2022-03-23 | 12000 | 0.15 | 23 |
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |

# Department

| D_ID | D_NAME |
|---|---|
| 20 | TRAINING |
| 21 | SALES |
| 22 | DIGITAL MARKETING |
| 23 | OPERATIONS |
| 24 | CREATIVE TEAM |
| 25 | BUssiness Dev |
| 26 | FINANCE |

# Product

| product_id | product_name | price | stock |
|---|---|---|---|
| 1 | Smartphone | 25000 | 100 |
| 2 | Laptop | 60000 | 50 |
| 3 | Tablet | 15000 | 75 |
| 4 | Headphones | 2000 | 200 |
| 5 | Smartwatch | 12000 | 40 |

# Order By Clause

- Order by clause is used to sort the data in ascending or in descending order.
- To sort the data in the descending order we need to use desc keyword.
- To sort the data in the ascending order we need to use asc keyword.
- Order by clause by default it will sort the data in ascending order if we are not mentioning which order to sort.
- **Syntax:**

  Order by column_name desc/asc;

## Queries on Order by clause

### 1. Write a query to display details of employee in the ascending order based on the employee first name.

**Solution:**

```
select *
from employee
order by e_fname asc;
```

**Output:**

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 8 | Aaron | Samuels | araon@gmail.com | 5423424 | 2014-06-24 | 12000 | 5.5 | 26 |
| 9 | Adam | Sam | adam@gmail.com | 325674 | 2015-04-16 | 53000 | 1.2 | 26 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 5 | John | Williams | john@gmail.com | 7456465 | 2010-12-15 | 18000 | 1.5 | 25 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 14 | Savitha | Singh | savi@gmail.com | 3443333 | 2022-03-23 | 12000 | 0.15 | 23 |
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |

Tap Academy

## 2. Write a query to display details of employee in the descending order based on the department id.

### Solution:

```
select *
from employee
order by d_id desc;
```

### Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 8 | Aaron | Samuels | araon@gmail.com | 5423424 | 2014-06-24 | 12000 | 5.5 | 26 |
| 9 | Adam | Sam | adam@gmail.com | 325674 | 2015-04-16 | 53000 | 1.2 | 26 |
| 5 | John | Williams | john@gmail.com | 7456465 | 2010-12-15 | 18000 | 1.5 | 25 |
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 14 | Savitha | Singh | savi@gmail.com | 3443333 | 2022-03-23 | 12000 | 0.15 | 23 |
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |

## 3. Write a query to display details of employee whose department id is greater than 20 and salary is greater than 10000, and display the result in the descending order based on last name.

### Solution:

```
select *
from employee
where d_id>20 and salary>10000
order by e_lname desc;
```

## Output:

| E_ID | E_FNAME | E_LNAME | E_EMAIL | E_PHONE_NO | HIRE_DATE | SALARY | COMMISSION_PCT | D_ID |
|---|---|---|---|---|---|---|---|---|
| 5 | John | Williams | john@gmail.com | 7456465 | 2010-12-15 | 18000 | 1.5 | 25 |
| 2 | William | taylor | william@gmail.com | 77665522 | 2012-05-25 | 45000 | 2 | 23 |
| 13 | Mamatha | T | mamatha@gmail.com | 53434334 | 2019-07-25 | 46000 | 1.2 | 21 |
| 1 | Michael | Smith | michael@gmail.com | 9997766 | 2012-03-29 | 34000 | 1 | 22 |
| 7 | Andy | Smith | andy@gmail.com | 8579452 | 2012-08-28 | 28500 | 0.9 | 22 |
| 14 | Savitha | Singh | savi@gmail.com | 3443333 | 2022-03-23 | 12000 | 0.15 | 23 |
| 8 | Aaron | Samuels | araon@gmail.com | 5423424 | 2014-06-24 | 12000 | 5.5 | 26 |
| 9 | Adam | Sam | adam@gmail.com | 325674 | 2015-04-16 | 53000 | 1.2 | 26 |
| 6 | Thomas | Raj | thomas@gmail.com | 9575673 | 2010-08-25 | 85000 | 0.6 | 26 |
| 3 | Richard | Miller | richard@gmail.com | 876297 | 2012-04-19 | 55000 | 0.4 | 24 |
| 11 | Madavan | Manish | madavan@gmail.com | 4353213 | 2016-12-30 | 25000 | 0.3 | 21 |
| 15 | Susan | Kumar | mru@gmail.com | 322378445 | 2023-04-30 | 19500 | 1.2 | 25 |
| 4 | Joseph | K | joseph@gmail.com | 66554433 | 2013-02-27 | 36000 | 0.5 | 22 |
| 12 | Jennifer | Anderson | jenni@gmail.com | 12342231 | 2018-09-27 | 45500 | 0.2 | 21 |
| 10 | Nancy | A | nancy@gmail.com | 3453424 | 2016-10-25 | 53000 | 1.2 | 22 |

## 4. Write a query to display the 2nd highest salary from employee table.

## Solution:

```
select salary
from employee
order by salary desc
limit 1 offset 1;
```

## Output:

| salary |
|---|
| 55000 |

**Note:** The limit is used to limit the number of records from the final result and it will be starting with 1. The offset is used to select the specific values from the column and it will be starting with 0.

Tap Academy

## 5. Write a query to display the top two salaries from the employee table.

### Solution1:

```
select salary
from employee
order by salary desc
limit 2 offset 0;
```

### Solution2:

```
select salary
from employee
order by salary desc
limit 2;
```

### Output:

```
+--------+
| salary |
+--------+
|  85000 |
|  55000 |
+--------+
```

**Note:** In the above solution1 limit 2 records and offset 0, it will select the values from 0 offset and from there it will limit 2 records hence we are getting the top two salaries.

Tap Academy

6. **Write a query to display the 2nd and 3rd highest salary from the employee table.**

**Solution:**

```
select salary
from employee
order by salary desc
limit 2 offset 1;
```

**Output:**

```
+--------+
| salary |
+--------+
|  55000 |
|  53000 |
+--------+
```

7. **Write a query to display the 2nd and 3rd least salary from the employee table.**

**Solution:**

```
select salary
from employee
order by salary asc
limit 2 offset 1;
```

**Output:**

```
+--------+
| salary |
+--------+
|  12000 |
|  18000 |
+--------+
```

Tap Academy

## 8. Write a query to display the Nth salary from the employee table.

### Solution:

```
select salary,dense_rank()
over(order by salary desc) as "6th highest salary"
from employee
limit 1 offset 6;
```

### Output:

```
+--------+---------------------+
| salary | 6th  highest salary |
+--------+---------------------+
|  45000 |                   6 |
+--------+---------------------+
```

### Note:

- Here dense_rank() is used to give the ranking to the sorted data and for each unique value it will give one number, and it will not skip any number, it will give ranking as shown below.

```
mysql> select salary,dense_rank()
    -> over(order by salary desc) as Ranking
    -> from employee;
+--------+---------+
| salary | Ranking |
+--------+---------+
|  85000 |       1 |
|  55000 |       2 |
|  53000 |       3 |
|  53000 |       3 |
|  46000 |       4 |
|  45500 |       5 |
|  45000 |       6 |
|  36000 |       7 |
|  34000 |       8 |
|  28500 |       9 |
|  25000 |      10 |
|  19500 |      11 |
|  18000 |      12 |
|  12000 |      13 |
|  12000 |      13 |
+--------+---------+
15 rows in set (0.00 sec)
```

Tap Academy

- Now by using the limit you can tell how many records you want and by using the offset you can tell which highest salary you want to display like $4^{th}$ , $5^{th}$ highest salary etc.
- Over() is used based on which column that you want to give the ranking, if you want highest you can sort that column in descending, if you want the least then you can sort that column in ascending order.

Tap Academy

# Functions

**Single Row functions**

- String Functions
- Math Functions
- Window Functions
- Date Functions
- Control Flow Functions
- Comparison Functions

**Multi Row functions**

- Aggregate Functions

**Note:** **Single row functions** are such functions which will return the result for each and every input/record. **Multi row functions** are such functions which will return the single output by combining all the given inputs/records.

## String Functions

### Types of String Functions

1. upper()
2. lower()
3. concat()
4. length()
5. substr()
6. instr()
7. replace()
8. trim()

Tap Academy

1. **upper():** The upper() function is used to convert the lowercase string or sequence of characters or character to uppercase.

**Example:**

```
select upper("this is upper function")
from dual;
```

**Output:**

```
+----------------------------------+
| upper("this is upper function") |
+----------------------------------+
| THIS IS UPPER FUNCTION           |
+----------------------------------+
1 row in set (0.00 sec)
```

---

2. **lower():** The lower() function is used to convert the uppercase strong or sequence of character or character to lowercase.

**Example:**

```
select lower("THIS IS LOWER FUNCTION")
from dual;
```

**Output:**

```
+----------------------------------+
| lower("THIS IS LOWER FUNCTION") |
+----------------------------------+
| this is lower function           |
+----------------------------------+
```

Tap Academy

**Note:** Here dual table is a default table which will be present in MySQL, no need to create this table you can just use this table to perform operations on any raw data.

---

**3.concat():** This concat() function is used to combine or join multiple strings or sequence of characters or characters as one.

**Syntax:**

Concat("value1","value2",....);

**Example:**

```
select concat("Tap","Academy")
from dual;
```

**Output:**

```
+-------------------------+
| concat("Tap","Academy") |
+-------------------------+
| TapAcademy              |
+-------------------------+
```

---

**4.length():** The length() function is used to fetch the length of the specified string or sequence of characters.

**Example:**

```
select length("Tap Academy")
from dual;
```

Tap Academy

**Output:**

```
+-----------------------+
| length("Tap Academy") |
+-----------------------+
|                    11 |
+-----------------------+
```

---

**5. substr():** The substr() function is used to fetch the part of the string.

**Syntax1:**

Substr(String, position);

**Example:**

```
select substr("TapAcademy",4)
from dual;
```

**Output:**

```
+-------------------------+
| substr("TapAcademy",4) |
+-------------------------+
| Academy                 |
+-------------------------+
```

**Note:** Position in MySQL starts from 1, it means from which position we want to fetch the data from the specified string.

**Syntax2:**

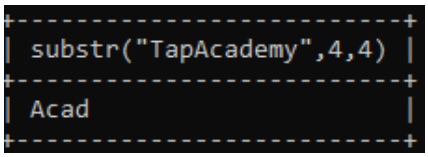
substr(String, position, number of characters);

**Example:**

```
select substr("TapAcademy",4,4)
from dual;
```

Tap Academy

**Output:**




**Note:** Number of characters means from the specified position how many characters we need to fetch.

---

**6. instr():** The instr() function is used to fetch the position of the character from the specified string.

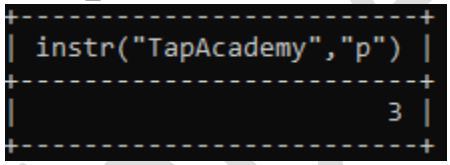
**Syntax:**

instr(String, character);

**Example:**

```
select instr("TapAcademy","p")
from dual;
```

**Output:**




---

**7. replace():** The replace() function is used to replace the data from the specified data.

**Syntax:**

replace(string, old_string, new_string)

**Example:**

```
select replace("tap Academy","t","T")
from dual;
```

Tap Academy

**Output:**

```
+--------------------------------+
| replace("tap Academy","t","T") |
+--------------------------------+
| Tap Academy                    |
+--------------------------------+
```

---

8. **trim():** The trim() function is used to truncate/remove/trim the data from the specified data. We need to use the **leading** keyword to trim the data from the beginning. We need to use **trailing** keyword to trim the data from the ending.

**Example1:**

```
select trim(leading “m” from “madam”)
from dual;
```

**Output:**

```
+-----------------------------------+
| trim(leading "m" from "madam") |
+-----------------------------------+
| adam                              |
+-----------------------------------+
```

**Example2:**

```
select trim(trailing “m” from “madam”)
from dual;
```

**Output:**

```
+------------------------------------+
| trim(trailing "m" from "madam") |
+------------------------------------+
| mada                               |
+------------------------------------+
```

Tap Academy

**Note:** We can use **ltrim()** function to remove the spaces from the left side of the string. We can use **rtrim()** function to remove the spaces from the right side of the string as show below:

## Example1:

```
select ltrim("          TapAcademy")
from dual;
```

**Output:**

```
+-----------------------------------+
| ltrim("             TapAcademy") |
+-----------------------------------+
| TapAcademy                        |
+-----------------------------------+
```

## Example2:

```
select rtrim("          TapAcademy      ")
from dual;
```

**Output:**

```
+-------------------------------------------+
| rtrim("             TapAcademy       ") |
+-------------------------------------------+
|               TapAcademy                  |
+-------------------------------------------+
```

TapAcademy

# Math Functions

## Types of Math Functions

1. **mod()**
2. **round()**
3. **truncate()**
4. **ceil()**
5. **floor()**

**1. mod():** The mod() function is used to get the remainder value after performing the division.

**Syntax:**

mod(dividend, divisor)

**Example:**

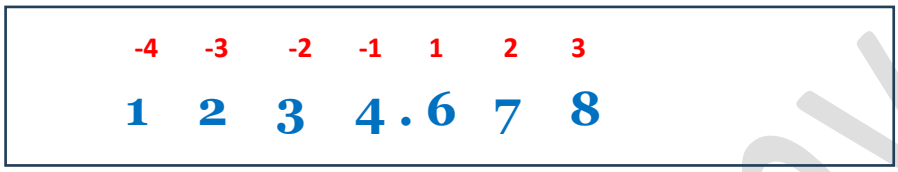
```
select mod(125,3)
from dual;
```

**Output:**

```
+------------+
| mod(125,3) |
+------------+
|          2 |
+------------+
```

Tap Academy

**2.round():** The round() function is used to round up the values at the specified position, here position can be positive or negative. After the decimal point we need to consider the positive positions, before the decimal point we need to consider the negative positions as show below.




## Let's take one example as show below:

```
select round(1234.567,2)
from dual;
```

In the above query they have mentioned as we need to round up the valued at the 2nd position. It's a positive value hence we need to check after the decimal point.

We have rounded the value at the 2nd position, once we have rounded we need to check whether the right number is greater than or equal to(>=) 5 or not as show below:

Tap Academy

If the right number is greater than or equal to 5 then the left number should be incremented by 1, in the above example right number is >=5 hence the left number will be incremented by 1 and it will become as 8 and it will return the result as 1234.68 as how below:

```
mysql> select round(1234.678,2)
    -> from dual;
+-------------------+
| round(1234.678,2) |
+-------------------+
|           1234.68 |
+-------------------+
```

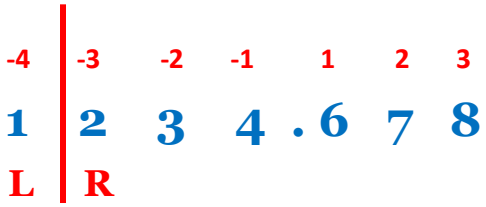
## Let's take one example for negative position as show below:

```
select round(1234.567,-3)
from dual;
```

In the above query they have mentioned as we need to round up the valued at the 3rd position. It's a negative value hence we need to check before the decimal point.

We have rounded the value at the 3rd position, once we have rounded, we need to check whether the right number is greater than or equal to (>=) 5 or not as show below:

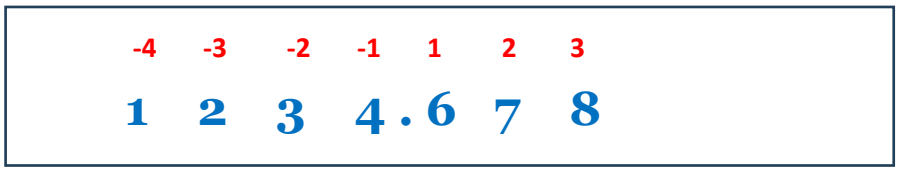
Tap Academy




If the right number is greater than or equal to 5 then the left number should be incremented by 1, in the above example right number is not >=5 hence the left number so it will not get incremented by 1 and it will be as 1 only, and after the left number how many numbers are present before the decimal point that number each will be converted into zero(0) and after the decimal point values will be ignored in the result as show below:

```
mysql> select round(1234.678,-3)
    -> from dual;
+--------------------+
| round(1234.678,-3) |
+--------------------+
|               1000 |
+--------------------+
```

**3. truncate():** The truncate() function is used to truncate/remove the values at the specified position, here position can be positive or negative. After the decimal point we need to consider the positive positions, before the decimal point we need to consider the negative positions as show below.




Tap Academy

## Let's take one example as show below:

```
select truncate(1234.678,2)
from dual;
```

In the above query they have mentioned as we need to truncate the valued at the 2nd position. It's a positive value hence we need to check after the decimal point.

Once we have truncated the value it will display the value directly, truncate will not check the right and left number and incrementation also will not happen in truncate as show below:

```
mysql> select truncate(1234.678,2)
    -> from dual;
+----------------------+
| truncate(1234.678,2) |
+----------------------+
|              1234.67 |
+----------------------+
```

## Let's take one example for negative position as show below:

```
select truncate(1534.567,-3)
from dual;
```

Tap Academy

In the above query they have mentioned as we need to truncate the valued at the 3rd position. It's a negative value hence we need to check before the decimal point.

We have truncated the value at the 3rd position, once we have truncated, it will directly take the value and it will display it will not check for the right and the left number and it will not increment the values, but after truncating the remaining each number will be converted as zero(0) and after decimal point the values will be ignored same as round() function as show below:

```
mysql> select truncate(1534.567,-3)
    -> from dual;
+-----------------------+
| truncate(1534.567,-3) |
+-----------------------+
|                  1000 |
+-----------------------+
```

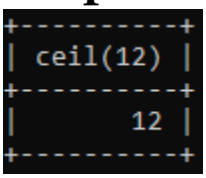
**4.ceil():** The ceil() function will return the integer value that is greater or equal too for the specified value as show below:

**Example1:**

```
select ceil(12)
from dual;
```

Tap Academy

**Output**

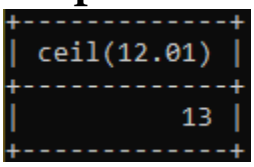


```
+----------+
| ceil(12) |
+----------+
|       12 |
+----------+
```

**Example2:**

```
select ceil(12.01)
from dual;
```

**Output**



```
+-------------+
| ceil(12.01) |
+-------------+
|          13 |
+-------------+
```

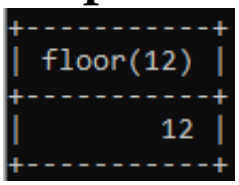
In the above we are using ceil() function so it will always give the integer value that is equal or greater than the specified values hence here we have 12.01 so it's rounding up the value and it is returning as 13.

5. **floor():** The floor() function will return the integer value that is less than or equal too for the specified value as show below:

**Example1:**

```
select floor(12)
from dual;
```

**Output**



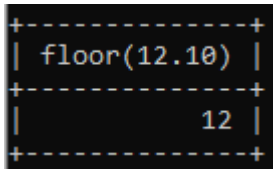
```
+-----------+
| floor(12) |
+-----------+
|        12 |
+-----------+
```

Tap Academy

## Example2:

```
select floor(12.10)
from dual;
```

## Output



```
+--------------+
| floor(12.10) |
+--------------+
|           12 |
+--------------+
```

In the above we are using floor() function so it will always give the integer value that is equal or less than the specified values hence here we have 12.10 so it's rounding up the value that is equal and it is returning as 12.

Tap Academy

# Date Functions

## Types of Date Functions

1. curdate()
2. now()
3. day()
4. month()
5. year()
6. datediff()
7. date_add()
8. last_day()
9. timestampdiff()
10. extract()
11. Date_format()

**1. curdate():** The curdate() function is used to display the current date that is present in your system.

**Example:**

```
select curdate();
```

**Output**

```
+------------+
| curdate()  |
+------------+
| 2023-09-19 |
+------------+
```

**Note:** Whenever we are using any inbuild function in date functions we no need to specify any table name in MySQL directly we can use the functions as shown above.

Tap Academy

**2. now():** The now() function will return the current date and time of the system.

**Example:**

```
select now();
```

**Output**

```
+---------------------+
| now()               |
+---------------------+
| 2023-09-19 15:31:03 |
+---------------------+
```

**Note:** This current date and time will be changed according the day you're executing this query, it won't be constant as shown above.

---

**3.now():** The now() function will return the current date and time of the system.

**Example:**

```
select now();
```

**Output**

```
+---------------------+
| now()               |
+---------------------+
| 2023-09-19 15:31:03 |
+---------------------+
```

Tap Academy

**4. day():** The day() function will return the day from the specified date.

**Example:**

```
select day("2023-09-19");
```

**Output**

```
+------------------+
| day("2023-09-19") |
+------------------+
|                19 |
+------------------+
```

**Note:** Give the same date format while executing the query, otherwise in MySQL it will not return result, instead it will return as NULL.

---

**5. month():** The month() function will return the month from the specified date.

**Example:**

```
select month("2023-09-19");
```

**Output**

```
+--------------------+
| month("2023-09-19") |
+--------------------+
|                  9 |
+--------------------+
```

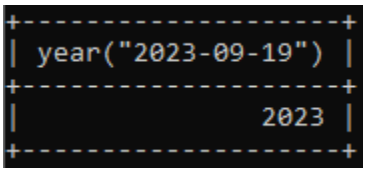
Tap Academy

**6.year():** The year() function will return the year from the specified date.

**Example:**

```
select year("2023-09-19");
```

**Output**



| year("2023-09-19") |
|---|
| 2023 |

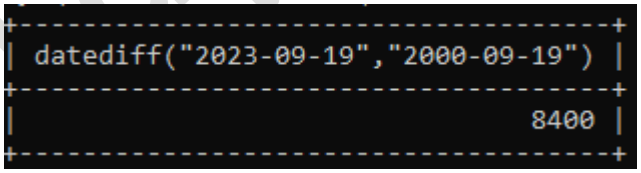
---

**7. datediff():** The datediff() function will return the difference between two specified dates in term of days.

**Example:**

```
select datediff("2023-09-19","2000-09-19");
```

**Output**



| datediff("2023-09-19","2000-09-19") |
|---|
| 8400 |

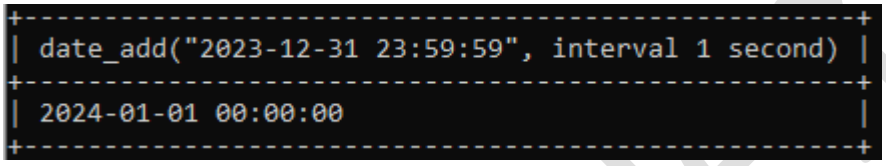
Tap Academy

**8. date_add:** The date_add() function will add the seconds to the given date.

**Example:**

```
select date_add("2023-12-31 23:59:59", interval 1 second);
```

**Output**



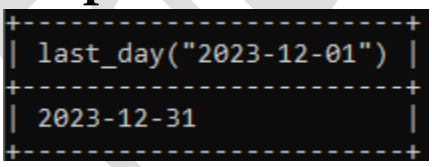

---

**9. last_day():** The last_day() function will return the last day of given date.

**Example:**

```
select last_day("2023-12-01");
```

**Output**




Tap Academy

**10. timestampdiff():** The timestampdiff() function is used to find the month,day,hours,minutes etc difference from the specified dates.

### Example1:

```
select timestampdiff(day,"2000-12-01" ,"2023-12-01");
```

### Output

```
+-----------------------------------------------+
| timestampdiff(day,"2000-12-01" ,"2023-12-01") |
+-----------------------------------------------+
|                                          8400 |
+-----------------------------------------------+
```

### Example2:

```
select timestampdiff(hour,"2000-12-01" ,"2023-12-01");
```

### Output

```
+------------------------------------------------+
| timestampdiff(hour,"2000-12-01" ,"2023-12-01") |
+------------------------------------------------+
|                                         201600 |
+------------------------------------------------+
```

Tap Academy

## Example3:

```
select timestampdiff(minute,"2000-12-01" ,"2023-12-01");
```

## Output

```
+-----------------------------------------------------+
| timestampdiff(minute,"2000-12-01" ,"2023-12-01") |
+-----------------------------------------------------+
|                                            12096000 |
+-----------------------------------------------------+
```

## Example4:

```
select timestampdiff(month,"2000-12-01" ,"2023-12-01");
```

## Output

```
+-----------------------------------------------------+
| timestampdiff(month,"2000-12-01" ,"2023-12-01") |
+-----------------------------------------------------+
|                                                 276 |
+-----------------------------------------------------+
```

## Example5:

```
select timestampdiff(year,"2000-12-01" ,"2023-12-01");
```

## Output

```
+-----------------------------------------------------+
| timestampdiff(year,"2000-12-01" ,"2023-12-01") |
+-----------------------------------------------------+
|                                                  23 |
+-----------------------------------------------------+
```

Tap Academy

11. **extract():** This function is used to extract the seconds, minutes,hours etc from the specified single date.

### Example1:

```
select extract(second from “2000-12-01 12:40:39” );
```

### Output

```
+-----------------------------------------------+
| extract(second from "2000-12-01 12:40:39" ) |
+-----------------------------------------------+
|                                            39 |
+-----------------------------------------------+
```

### Example2:

```
select extract(minute from “2000-12-01 12:40:39” );
```

### Output

```
+-----------------------------------------------+
| extract(minute from "2000-12-01 12:40:39" ) |
+-----------------------------------------------+
|                                            40 |
+-----------------------------------------------+
```

Tap Academy

## Example3:

```
select extract(hour from “2000-12-01 12:40:39” );
```

## Output

```
+-----------------------------------------+
| extract(hour from "2000-12-01 12:40:39" ) |
+-----------------------------------------+
|                                      12 |
+-----------------------------------------+
```

---

**12. date_format():** This function is used to display the date in the specified format or way.

## Consider the below table to understand the formats:

| Specifier | Meaning |
|---|---|
| %a | Three-characters abbreviated weekday name e.g., Mon, Tue, Wed, etc. |
| %b | Three-characters abbreviated month name e.g., Jan, Feb, Mar, etc. |
| %c | Month in numeric e.g., 1, 2, 3...12 |
| %D | Day of the month with English suffix e.g., 0th, 1st, 2nd, etc. |
| %d | Day of the month with leading zero if it is 1 number e.g., 00, 01,02, ...31 |

| Specifier | Meaning |
|---|---|
| %e | Day of the month without leading zero e.g., 1,2,...31 |
| %f | Microseconds in the range of 000000..999999 |
| %H | Hour in 24-hour format with leading zero e.g., 00..23 |
| %h | Hour in 12-hour format with leading zero e.g., 01, 02...12 |
| %I | Same as %h |
| %i | Minutes with leading zero e.g., 00, 01,...59 |
| %j | Day of year with leading zero e.g., 001,002,...366 |
| %k | Hour in 24-hour format without leading zero e.g., 0,1,2...23 |
| %l | Hour in 12-hour format without leading zero e.g., 1,2...12 |
| %M | Full month name e.g., January, February,...December |
| %m | Month name with leading zero e.g., 00,01,02,...12 |
| %p | AM or PM, depending on other time specifiers |
| %r | Time in 12-hour format hh:mm:ss AM or PM |
| %S | Seconds with leading zero 00,01,...59 |
| %s | Same as %S |
| %T | Time in 24-hour format hh:mm:ss |

| Specifier | Meaning |
|---|---|
| %U | Week number with leading zero when the first day of week is Sunday e.g., 00,01,02...53 |
| %u | Week number with leading zero when the first day of week is Monday e.g., 00,01,02...53 |
| %V | Same as %U; it is used with %X |
| %v | Same as %u; it is used with %x |
| %W | Full name of weekday e.g., Sunday, Monday,..., Saturday |
| %w | Weekday in number (0=Sunday, 1= Monday,etc.) |
| %X | Year for the week in four digits where the first day of the week is Sunday; often used with %V |
| %x | Year for the week, where the first day of the week is Monday, four digits; used with %v |
| %Y | Four digits year e.g., 2000 and 2001. |
| %y | Two digits year e.g., 10,11,and 12. |
| %% | Add percentage (%) character to the output |

# Consider the below table to understand the combination of formats:

| DATE_FORMAT string | Formatted date |
|---|---|
| %Y-%m-%d | 2013-07-04 |
| %e/%c/%Y | 4/7/2013 |
| %c/%e/%Y | 7/4/2013 |
| %d/%m/%Y | 4/7/2013 |
| %m/%d/%Y | 7/4/2013 |
| %e/%c/%Y %H:%i | 4/7/2013 11:20 |
| %c/%e/%Y %H:%i | 7/4/2013 11:20 |
| %d/%m/%Y %H:%i | 4/7/2013 11:20 |
| %m/%d/%Y %H:%i | 7/4/2013 11:20 |
| %e/%c/%Y %T | 4/7/2013 11:20 |
| %c/%e/%Y %T | 7/4/2013 11:20 |
| %d/%m/%Y %T | 4/7/2013 11:20 |
| %m/%d/%Y %T | 7/4/2013 11:20 |
| %a %D %b %Y | Thu 4th Jul 2013 |
| %a %D %b %Y %H:%i | Thu 4th Jul 2013 11:20 |

| DATE_FORMAT string | Formatted date |
|---|---|
| %a %D %b %Y %T | Thu 4th Jul 2013 11:20:05 |
| %a %b %e %Y | Thu Jul 4 2013 |
| %a %b %e %Y %H:%i | Thu Jul 4 2013 11:20 |
| %a %b %e %Y %T | Thu Jul 4 2013 11:20:05 |
| %W %D %M %Y | Thursday 4th July 2013 |
| %W %D %M %Y %H:%i | Thursday 4th July 2013 11:20 |
| %W %D %M %Y %T | Thursday 4th July 2013 11:20:05 |
| %l:%i %p %b %e, %Y | 7/4/2013 11:20 |
| %M %e, %Y | 4-Jul-13 |
| %a, %d %b %Y %T | Thu, 04 Jul 2013 11:20:05 |

## Example:

```
select date_format("2023-09-19"," %W %D %M %Y");
```

## Output

```
+-----------------------------------------+
| date_format("2023-09-19"," %W %D %M %Y") |
+-----------------------------------------+
|  Tuesday 19th September 2023            |
+-----------------------------------------+
```

# Comparison Functions

## Types of Comparison Functions

1. **isnull()**
2. **coalesce()**

1. **isnull():** This function take only one argument/parameter, if the argument is null it will return 1, else it will return 0.

**Example1:**

```
select isnull(null);
```

**Output**

```
+--------------+
| isnull(null) |
+--------------+
|            1 |
+--------------+
```

**Example2:**

```
select isnull(1);
```

**Output**

```
+------------+
| isnull(1) |
+------------+
|          0 |
+------------+
```

Tap Academy

2. **coalesce():** This function takes two arguments, here if first argument is null, it will replace null with the second argument value and it will display that, if first argument is not null it will display the first argument only.

### Example1:

```
select coalesce(null,1);
```

### Output

```
+------------------+
| coalesce(null,1) |
+------------------+
|                1 |
+------------------+
```

### Example2:

```
select coalesce(2,1);
```

### Output

```
+---------------+
| coalesce(2,1) |
+---------------+
|             2 |
+---------------+
```

Tap Academy

# Control Flow Functions

## Types of Control Flow Functions

1. **case**
2. **if**
3. **ifnull**
4. **nullif**

1. **case:** MySQL case expression is a control flow structure that allows you to add if-else logic to a query.

### Syntax

```
CASE value
  WHEN value1 THEN result1
  WHEN value2 THEN result2
  ...
  [ELSE else_result]
END
```

### Example

```
select salary,
case
when e_fname=”William” then salary*10/100
when e_fname=”Aaron” then salary*15/100
else salary*20/100
end
from employee;
```

Tap Academy

**Note:** In the above query we are checking multiple conditions and for each condition we are fetching the separate data. So here if the employee first name is **William** it will fetch the **10%** of the salary of William, if the employee first_name is **Aaron** it will fetch the **15%** of the salary of Aaron, from the remaining employees it will fetch **20%** of their salary. Hence whenever, we want to check multiple condition based on the conditions if we want to fetch separate data then we can use **CASE** function.

---

**2. if:** MySQL IF function is one of the MySQL control flow functions that returns a value based on a condition.

## Syntax

```
IF(expr,if_true_expr,if_false_expr)
```

## Example

```
select if(5=5,'true','false');
```

## Output

```
+------------------------+
| if(5=5,'true','false') |
+------------------------+
| true                   |
+------------------------+
```

**Note:** In the above query we are checking the condition and using IF, i.e., if 5=5 then it will return true else it will return false.

Tap Academy

**3.ifnull:** MySQL ifnull function is one of the MySQL control flow functions that accepts two arguments and returns the first argument if it is not null. Otherwise, the ifnull function returns the second argument.

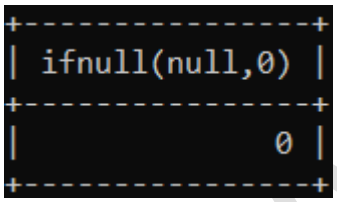
## Syntax

IFNULL(expression_1,expression_2);

## Example

```
select ifnull(null,0) ;
```

## Output




---

**4.nullif:** The nullif function is one of the control flow functions that accepts 2 arguments. The nullif function returns null if the first argument is equal to the second argument, otherwise it returns the first argument.

## Syntax

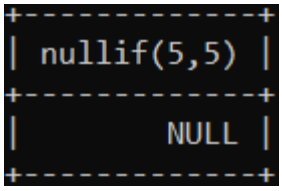
NULLIF(expression_1,expression_2);

## Example

```
select nullif(5,5);
```

Tap Academy

## Output



| nullif(5,5) |
|---|
| NULL |

---

# Window Functions

MySQL has supported window functions since version 8.0. The window functions allow you to solve query problems in new, easier ways and with better performance.

## Syntax:

```
window_function_name(expression) OVER (
  [partition_defintion]
  [order_definition]
  [frame_definition]
)
```

## Note:

- First, specify the window function name followed by an expression.
- Second, specify the OVER clause which has three possible elements: partition definition, order definition, and frame definition.
- The opening and closing parentheses after the OVER clause are mandatory, even with no expression.
- The partition_clause breaks up the rows into chunks or partitions.
- The ORDER BY clause specifies how the rows are ordered within a partition.
- A frame clause is a subset of the current partition.

Tap Academy

## Types of Window Functions

1. **rank()**
2. **dense_rank()**
3. **row_number()**

1. **rank():** It will assign the rank to the values in the specified column, but if same value is present the rank value can be skipped if we are using rank() function to give the rank.

### Syntax:

```
select salary, rank()
over(order by salary desc)
from employee;
```

### Output:

| salary | rank() over(order by salary desc) |
|---|---|
| 85000 | 1 |
| 55000 | 2 |
| 53000 | 3 |
| 53000 | 3 |
| 46000 | 5 |
| 45500 | 6 |
| 45000 | 7 |
| 36000 | 8 |
| 34000 | 9 |
| 28500 | 10 |
| 25000 | 11 |
| 19500 | 12 |
| 18000 | 13 |
| 12000 | 14 |
| 12000 | 14 |

**Note:** In the above example we are sort the salary in descending order and we are giving the ranks for each value, here you can see that 4th rank has been skipped, so if we are using the rank() function we can give the ranks to the values but there will be a chance that ranks can be skipped as shown above.

**2. dense_rank():** Assigns a rank to every row within its partition based on the ORDER BY clause. It assigns the same rank to the rows with equal values. If two or more rows have the same rank, then there will be no gaps in the sequence of ranked values.

## Syntax:

```
select salary, dense_rank()
over(order by salary desc)
from employee;
```

## Output:

| salary | dense_rank() over(order by salary desc) |
|---|---|
| 85000 | 1 |
| 55000 | 2 |
| 53000 | 3 |
| 53000 | 3 |
| 46000 | 4 |
| 45500 | 5 |
| 45000 | 6 |
| 36000 | 7 |
| 34000 | 8 |
| 28500 | 9 |
| 25000 | 10 |
| 19500 | 11 |
| 18000 | 12 |
| 12000 | 13 |
| 12000 | 13 |

## 3.row_number():

MySQL introduce the row_number() function since version 8.0. The row_number() is a window function that assigns a sequential number to each row in the result set. The first number begins with one.

### Syntax:

```
select salary, dense_rank()
over(order by salary desc)
from employee;
```

### Output:

| salary | row_number() over(order by salary desc) |
|---|---|
| 85000 | 1 |
| 55000 | 2 |
| 53000 | 3 |
| 53000 | 4 |
| 46000 | 5 |
| 45500 | 6 |
| 45000 | 7 |
| 36000 | 8 |
| 34000 | 9 |
| 28500 | 10 |
| 25000 | 11 |
| 19500 | 12 |
| 18000 | 13 |
| 12000 | 14 |
| 12000 | 15 |

# Aggregate Functions

Aggregate functions are also called as group functions or multi row functions. This functions always return single output only.

## Types of Aggregate Functions

1. **count()**
2. **sum()**
3. **min()**
4. **max()**
5. **avg()**

1. **count():** It will return the count of the values in the specified column or data.

### Example

```
select count(*)
from employee;
```

### Output

```
+----------+
| count(*) |
+----------+
|       15 |
+----------+
```

Tap Academy

**2.sum():** It will return the sum of the values of the specified column or data.

## Example

```
select sum(salary)
from employee;
```

## Output

```
+-------------+
| sum(salary) |
+-------------+
|      567500 |
+-------------+
```

---

**3.min():** It will return the minimum value of the specified column or data.

## Example

```
select min(salary)
from employee;
```

## Output

```
+-------------+
| min(salary) |
+-------------+
|       12000 |
+-------------+
```

Tap Academy

**4. max():** It will return the maximum value of the specified column or data.

## Example

```
select max(salary)
from employee;
```

## Output

```
+-------------+
| max(salary) |
+-------------+
|       85000 |
+-------------+
```

---

**5. avg():** It will return the average value of the specified column or data.

## Example

```
select avg(salary)
from employee;
```

## Output

```
+--------------------+
| avg(salary)        |
+--------------------+
| 37833.333333333336 |
+--------------------+
```

Tap Academy

# Queries on Functions

1. **Write a query to display the first name and last name of an employee from the employee table as a full name.**

**Solution:**

```
select concat(e_fname," ",e_lname) as "full name"
from employee;
```

**Output:**

```
+--------------------+
| full name          |
+--------------------+
| Michael Smith      |
| William taylor     |
| Richard Miller     |
| Joseph K           |
| John Williams      |
| Thomas Raj         |
| Andy Smith         |
| Aaron Samuels      |
| Adam Sam           |
| Nancy A            |
| Madavan Manish     |
| Jennifer Anderson  |
| Mamatha T          |
| Savitha Singh      |
| Susan Kumar        |
+--------------------+
```

Tap Academy

**Note:** Whatever we give in the select statement it will be display same as a column name in the result, so whatever the operations we perform it will display in the result hence we need to give proper and meaningful name for the column whenever we are displaying the result, to give that meaning full name for columns we need to use the **column alias**.

# Column Alias

Column alias is a temporary name which is given for the columns while writing the query to display the meaningful column names.

## Syntax:

Column_name as “column alias name”

## Example

```
select concat(e_fname,” “,e_lname) as “full name”
from employee;
```

## Output

```
+--------------------+
| full name          |
+--------------------+
| Michael Smith      |
| William taylor     |
| Richard Miller     |
| Joseph K           |
| John Williams      |
| Thomas Raj         |
| Andy Smith         |
| Aaron Samuels      |
| Adam Sam           |
| Nancy A            |
| Madavan Manish     |
| Jennifer Anderson  |
| Mamatha T          |
| Savitha Singh      |
| Susan Kumar        |
+--------------------+
```

## Note:

- The keyword “as” is optional we can use it or we can skip it as well but as a standard syntax we can include that keyword while giving column alias as it will be more readable and understandable.
- Giving column alias in double quotes is mandatory if we are giving more than one word for the alias name else it’s not mandatory.

Tap Academy

2. **Write a query to display the length of all the department names from the department table.**

**Solution:**

```
select length(d_name)
from department;
```

**Output:**

```
+----------------+
| length(d_name) |
+----------------+
|              8 |
|              5 |
|             17 |
|             10 |
|             13 |
|             13 |
|              7 |
+----------------+
```

3. **Write a query to display "ill" from the employee first name whose first name is William**

**Solution:**

```
select substr(e_fname,2,3)
from employee
where e_fname="William";
```

**Output:**

```
+---------------------+
| substr(e_fname,2,3) |
+---------------------+
| ill                 |
+---------------------+
```

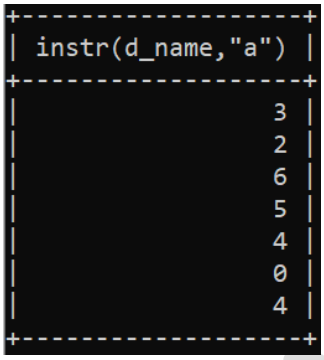
Tap Academy

## 4. Write a query to display the position of “a” character from the department

### Solution:

```
select instr(d_name,”a”)
from department;
```

### Output:



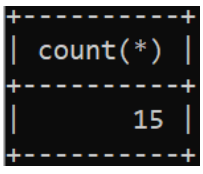
| instr(d_name,"a") |
|---|
| 3 |
| 2 |
| 6 |
| 5 |
| 4 |
| 0 |
| 4 |

## 5. Write a query to display the count of employees who are working in a company.

### Solution1:

```
select count(*)
from employee;
```

### Output:



| count(*) |
|---|
| 15 |

Tap Academy

### Solution2:

```
select count(e_id)
from employee;
```

### Output:

```
+-------------+
| count(e_id) |
+-------------+
|          15 |
+-------------+
```

## 6. Write a query to display sum of all salaries from the employee table

### Solution:

```
select sum(salary)
from employee;
```

### Output:

```
+-------------+
| sum(salary) |
+-------------+
|      567500 |
+-------------+
```

Tap Academy

## 7. Write a query to display the salary of an employee who is earning less than all employees.

### Solution:

```
select min(salary)
from employee;
```

### Output:

```
+-------------+
| min(salary) |
+-------------+
|       12000 |
+-------------+
```

## 8. Write a query to display the salary of an employee who is earning more than all employees.

### Solution:

```
select max(salary)
from employee;
```

### Output:

```
+-------------+
| max(salary) |
+-------------+
|       85000 |
+-------------+
```

Tap Academy

## 9. Write a query to display average salary from the employee table.

### Solution:

```
select avg(salary)
from employee;
```

### Output:

```
+--------------------+
| avg(salary)        |
+--------------------+
| 37833.333333333336 |
+--------------------+
```

Tap Academy